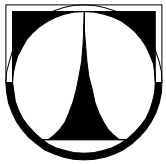
TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií



DIPLOMOVÁ PRÁCE

Liberec 2011 **Bc. Lukáš Vančura**

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 3902T005 – Automatické řízení a inženýrská informatika

Software pro analýzu záznamů měření elektrických veličin

Software for analysis of archives of electrical measurements

Diplomová práce

Autor: **Bc. Lukáš Vančura**
Vedoucí práce: Ing. Jan Kraus
Konzultant: Ing. Tomáš Tobiška

V Liberci 10. 5. 2011

Zde bude vloženo zadání DP

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Abstrakt

Cílem diplomové práce je vytvoření prostředků, umožnující automatickou analýzu naměřených dat. K tomuto účelu slouží algoritmy vytvářející optimalizované modely dat a algoritmy vyhledávající v datech opakující se úseky. Závěrečnou částí je tvorba klientské aplikace implementující předešlé prostředky.

Pro účely modelování dat je využito matematické knihovny Math.NET, s jejímž použitím se problematika vytváření optimálních modelů soustředí především na řešení optimalizační úlohy. K tomuto účelu je využito genetického algoritmu. Algoritmy zajišťující vyhledávání shodných úseků pracují na principu popisu dat určitými parametry, na jejichž základě se rozhoduje o shodě. Klientská aplikace doplňuje zmíněné algoritmy o uživatelské rozhraní, pomocí kterého je možné nastavovat různé parametry a zobrazovat výsledky.

Během diplomové práce vznikla řada algoritmů na modelování dat, z nichž většina využívá právě uvedené genetické optimalizace. Dále bylo vytvořeno několik prostředků na vyhledávání a porovnávání úseků dat, přičemž je využito dvou rozdílných přístupů k vyhodnocení shody mezi signály. Po úspěšném dokončení klientské testovací aplikace se potvrdilo, že většina vytvořených algoritmů, jak pro modelování dat, tak pro vyhledávání shodných úseků, úspěšně funguje. Nabízí se otázka, zda některé prostředky vytvořené v této diplomové práci najdou svoje uplatnění.

Klíčová slova: model, optimalizace, vyhledávání, uživatelské rozhraní, testování

Abstract

Graduation thesis deal with automatic analysis of archives of electrical measurements. For this purposes serves algorithms which creates optimized models of measured data and algorithms for searching identical and repeating samples of signal. Last part of this thesis is oriented on creating a client application, which uses all created algorithms.

Models of data are created by using math library Math.NET. Therefore main problem for solution is an optimization, which serves for creating models with minimal error against original data. Genetic algorithm is used for this purpose. Group of algorithms for searching identical and repeating samples of signal are based on describing a signal with parameters, which are used for decision about similarity of the signals. Client application adds a user interface to algorithms, which serves for setting up some parameters and for displaying of results.

Most of algorithms for modeling of signal created during development of this thesis uses genetic algorithm. Functions for searching identical signals are based on two different methods of comparing of signals. After finishing development of client application and after testing all functions was found that most of algorithms works correctly. At this moment I don't know, whether some algorithms created in this graduation thesis will be used sometime in real applications.

Keywords: model, optimization, searching, user interface, testing

Obsah

1 Úvod

Předmětem mé diplomové práce je návrh metod, které určitým způsobem modelují zadaný průběh signálu, tvorba algoritmů pro detekci specifických opakujících se jevů a tvorba klientské aplikace s využitím vytvořených prostředků. Modelování průběhu signálu má za účel předzpracování dat před následným využitím dalších algoritmů. Klientská aplikace slouží k otestování a vyhodnocení dosažených cílů.

Problematice modelování signálu je věnována velká část diplomové práce, jelikož se jedná a zajímavý a rozsáhlý problém. Hned na úvod by bylo dobré ozřejmit, co se pod tímto pojmem skrývá. V prvé řadě není účelem mé diplomové práce vytváření matematických modelů signálů. Modelování zde představuje výběr určitého počtu bodů ze zadaného průběhu a jejich následné proložení určitým typem křivky. Pro problematiku proložení bodů křivkou je využito již hotového nástroje. Tímto nástrojem je míněna matematická knihovna Math.NET. Hlavním cílem modelování je tedy výběr optimálních bodů, které se následně s využitím zmíněné knihovny proloží zadaným typem křivky. Cílem je pochopitelně vybrat takové body, aby se modelovaný průběh co nejvíce podobal průběhu originálnímu. Je tedy zřejmé, že je nutné využít nějakého druhu optimalizačních algoritmů, které budou chybu modelu minimalizovat. Pro účely optimalizace je v diplomové práci využito principů genetického algoritmu. Účelem práce není přesná implementace daného prostředku, ale s využitím jeho základních principů navrhnout vlastní optimalizační postupy. Cílem modelování se tak stává výběr optimálních bodů ze zadaného průběhu a předání těchto bodů matematické knihovně. Výsledkem by měl být co nejvěrněji namodelovaný průběh.

Pro úspěšnou klasifikaci opakujících se jevů v datech je nutné nejprve dané jevy nalézt. Vytvoření vyhledávacích a porovnávacích algoritmů je tak druhou částí diplomové práce. Účelem je opět návrh vlastních metod a postupů řešení dané problematiky. Tato část diplomové práce se tak zabývá především způsobem, jakými parametry definovat rozdíl mezi signály, případně jakým způsobem určité průběhy specifikovat, aby bylo možné je následně porovnávat. Cílem je tedy výběr těchto parametrů a jejich implementace do algoritmů. Výsledkem by měly být takové prostředky, které umožní v rozsáhlých datech nalézt požadované opakující se úseky.

Poslední částí diplomové práce je tvorba klientské aplikace implementující výše uvedené postupy. Následně její použití k otestování dosažených výsledků. Z důvodu využití aplikace pro testovací účely je nutné mimo nasazení modelovacích

a vyhledávacích algoritmů zajistit také získání potřebných údajů k vyhodnocení jednotlivých postupů. Jedním z účelů poslední části diplomové práce je nasazení vytvořených algoritmů do reálné aplikace. Je to dáno tím, že vyvíjené algoritmy jsou pouze funkce s určitými vstupními parametry. Nasazení těchto algoritmů v klientské aplikaci znamená řešit zadávání vstupních parametrů a zobrazování výsledků. Jedná se tedy o vyzkoušení a ukázku možného reálného využití v praxi. Výsledkem by měla být uživatelská aplikace, využívající vytvořené algoritmy na modelování dat a vyhledávání opakujících se úseků. V neposlední řadě i zhodnocení dosažených výsledků.

2 Přehled použitých prostředků

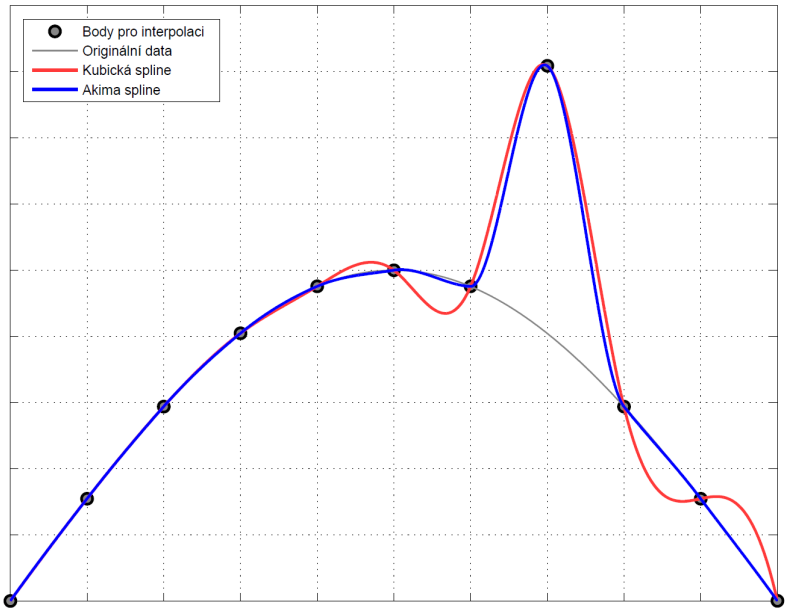
2.1 Modelování dat

Modelování dat má za účel popsat velké množství vstupních dat pomocí menšího množství hodnot a parametrů. Hodnotami jsou rozuměny body, kterými je proložena určitá křivka. Parametrem je poté právě druh této křivky. Cílem je, pomocí přiměřeného počtu hodnot proložených vhodnou křivkou, co nejlépe interpolovat skutečná data. K porovnání kvality modelu, tedy jak věrně model kopíruje originální data, je využito kvadratického kritéria.

2.1.1 Druhy interpolací

Pro interpolaci dat je využito knihovny Math.NET Numerics pro C#. Knihovna obsahuje několik typů interpolačních křivek. Použití knihovny je jednoduché, vstupem jsou hodnoty na ose X, dále hodnoty na ose Y. Některé typy interpolací vyžadují navíc třetí seznam hodnot. Může se jednat o derivace v jednotlivých bodech, případně o váhy jednotlivých bodů. Vhledem k uvedenému způsobu užívání funkcí pro modelování průběhu není nutné znát k jednotlivým interpolacím přesně princip jejich činnosti. Krátký popis k jednotlivým druhům interpolací, obsažených ve zmíněné knihovně, jsem však do textu diplomové práce zařadil. Uvedený text je převzat z [2], kde je možné se v anglickém znění dočíst o uvedených interpolacích více.

Akima spline interpolace



Obr. 2.1: Porovnání kubického a Akima spline (převzato z [2])

Tato interpolace je speciální spline, u které je hlavní předností stabilita poblíž bodů výrazně vzdálenějších od ostatních. Rozdíl je nejvíce vidět na obrázku 2.1 v porovnání s interpolací pomocí kubického spline. Další zajímavou vlastností této interpolace je výpočet interpolovaných hodnot mezi body v intervalu $[X_i, X_{i+1}]$, které závisí pouze na f_{i-2}, f_{i-1}, f_i, f_{i+1} f_{i+2} a f_{i+3}. Také je nutné zmínit, že je tato interpolace nelineární, což znamená, že interpolace součtu dvou funkcí není rovna součtu interpolací jednotlivých funkcí.

Barycentrická interpolace

Jedná se o vylepšenou racionální interpolaci. Tento algoritmus využívá barycentrické reprezentace racionální interpolace. Po určení váhových hodnot w_j se jedná o racionální funkci s jmenovatelem M-tého stupně.

Hermitova kubická spline interpolace

Interpolovaná křivka je spline třetího řádu. Oproti klasické kubické spline interpolaci využívá navíc první derivace v bodech určujících interpolovaný průběh. Výsledkem je spline, který má spojitou první derivaci. Druhá derivace již však spojitá není.

Kubická spline interpolace

Kubický spline je definován funkčními hodnotami v uzlech a hodnotami derivace na okrajích interpolovaného úseku. V případě, kdy jsou známy přesné hodnoty prvních derivací na hranicích interpolovaného úseku, se jedná o základní spline. Pokud hodnoty derivací známy nejsou, lze je nahradit hodnotami rovnými nule. Poté se jedná o takzvaný přirozený spline. Přesnost uvedené interpolace je větší uprostřed interpolovaného úseku. Směrem k jeho hranicím přesnost interpolace klesá. V případě, kdy je známa derivace jen na jedné hranici, je možné kombinovat základní a přirozený spline.

Ekvidistantní polynomická interpolace

Polynomická interpolace je výhodná především díky své jednoduchosti a dobrým výsledkům. V současné době se však více uplatňují jiné typy interpolací, a to spline a racionální funkce. Nicméně i přesto je polynomiální interpolace stále jedním z nástrojů numerické analýzy.

Floater Hormannova racionální interpolace

Floater Hormannova interpolatce je dána racionální funkcí. Vlastnosti zmíněné interpolace jsou rychlost, stabilita a spolehlivost. V mnoha ohledech je podobná interpolaci pomocí spline. Stupeň interpolace je označován písmenem d a platí $0 \leq d \leq n$. Písmenem n je označen počet bodů v interpolovaném úseku.

Lineární spline interpolace

Interpolace tímto typem křivky je po částech lineární funkce. Nevyniká příliš vysokou přesností a její první derivace je nespojitá. Nicméně v některých případech může být výhodnější použít právě lineární interpolaci namísto spline s vyšším stupněm. Lineární spline také zachovává monotónnost souboru bodů.

Nevillova polynomická interpolace

Jedná se o speciální polynom řádu nejvýše n. Pro interpolovaný úsek poté platí, že obsahuje n + 1 bodů. Nevillův interpolační algoritmus je založen na Newtonově formě interpolačního polynomu a rekurzivním vztahu pro dělené diference.

Spline interpolace

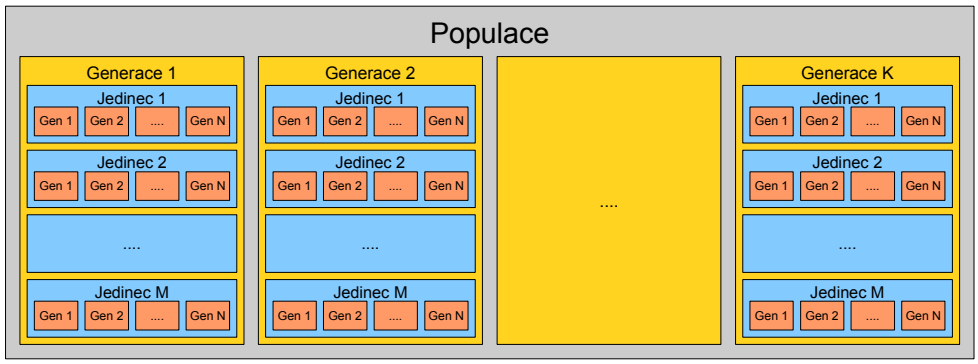
Princip interpolace pomocí spline je následující. Celý interpolační interval je rozdělen na části. Každá část je interpolována s polynomem třetího stupně. Hodnoty koeficientů polynomu jsou určeny v závislosti na interpolační metodě. Musí však být zaručena spojitost interpolovaného průběhu a také průchod všemi zadanými body. Výhodami jsou stabilita a výpočetní jednoduchost. Výpočet probíhá řešením soustavy lineárních rovnic.

2.2 Optimalizace interpolovaného průběhu

Použitím knihovny Math.NET a jejich funkcí na interpolování průběhů se naskýtá problém, jaká data do zmíněných funkcí zadat. Samozřejmostí je požadavek na co nejvěrnější interpolování originálního průběhu. Přesnost interpolace lze zvyšovat počtem bodů, které do interpolačních funkcí vstupují, nicméně je potřeba tyto body vybrat. Pro výběr optimálních bodů charakterizujících výsledný interpolovaný průběh je potřeba řešit optimalizační úlohu. Kvalita interpolace je určena kvadratickou odchylkou, což je funkce n neznámých, kde n udává počet bodů pro interpolaci. Tato funkce je však neznámá, a proto použití běžných metod, jako je např. Newtonova metoda, není možné. Pro takto složité úlohy se proto nabízí možnost použití genetického algoritmu.

2.2.1 Genetický algoritmus

Princip tohoto algoritmu je podobný genetice v biologii a odtud také pramení jeho název genetický algoritmus. Ve své podstatě se jedná o vývoj populace, která je tvořena jedinci. Ty se navzájem kříží nebo mutují a vznikají tak noví jedinci. Domnívám se, že před popisem samotného algoritmu bude vhodné uvést několik pojmů, které s touto problematikou souvisejí. Mnohem podrobněji se lze s touto problematikou seznámit např. v [8], odkud jsem nastudoval dané téma.

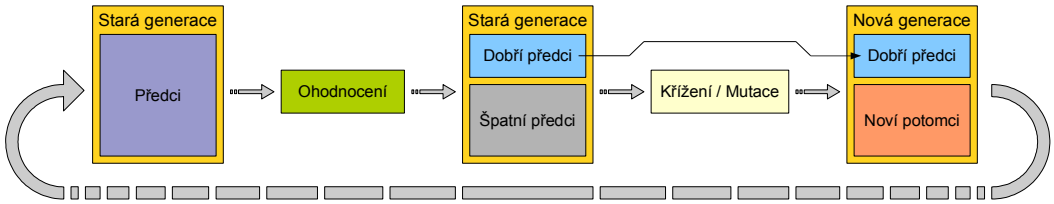


Obr. 2.2: Zobrazení hierarchie využívané v genetickém algoritmu

Nejširším pojmem je populace, ve které se odehrává celá optimalizace. Pokud by se mělo opět použít přirovnání k biologii, potom by populaci tvořil určitý živočišný druh. Populaci tvoří postupně jednotlivé generace. V genetickém algoritmu odpovídá jedna generace jednomu kroku iteračního výpočtu. Podobnost s reálným světem je v tomto případě trochu odlišná, neboť v genetickém algoritmu neexistuje více generací v jeden okamžik. Generace je tvořena určitým počtem jedinců a každý jedinec reprezentuje jedno konkrétní řešení optimalizované úlohy. Podle kvality řešení, které daný jedinec reprezentuje, je poté možné posoudit jeho zdatnost. Ta slouží k výběru jedinců, ze kterých vznikají potomci. Každého jedince tvoří určitý počet genů, které vypovídají o jeho vlastnostech. Pro lepší názornost jsem zařadil obrázek 2.2.

Po přiblížení nejzákladnější terminologie je možné přejít k vysvětlení, jak genetický algoritmus pracuje. Celý proces je z nejširšího pohledu možné rozdělit do tří částí. Nejdříve se musí vytvořit první generace, která se zpravidla generuje zcela náhodně. Poté proběhne iteračním výpočtem vývoj populace a nakonec se celá optimalizační úloha ukončí a určí se finální řešení. Ukončení výpočtů je možné více způsoby. Typickým příkladem může být pevně stanovený počet iteračních kroků, nebo ukončení v okamžiku, kdy se několik generací po sobě nelepší.

Nejzajímavější částí genetického algoritmu je postupný vývoj populace. Jak bylo zmíněno výše, jedná se o iterační výpočet, kdy ze staré generace v jednom kroku vzniká generace nová. Během tohoto jednoho kroku se však provádí několik procesů. V první řadě se jedná o ohodnocení jedinců v populaci. Ohodnocení vyjadřuje zdatnost jedince, neboli kvalitu řešení, jenž příslušný jedinec reprezentuje. Následuje křížení a mutování jedinců ve staré generaci a spolu s tím vznik nové. Je vhodné část nejlepších jedinců ze staré generace přímo zahrnout do nové, aby nenastal případ, že nová generace bude horší než stará. Vytvořením kompletní nové generace končí jeden krok genetického algoritmu. Princip činnosti je znázorněn na obrázku 2.3.



Obr. 2.3: Princip vzniku nových generací

2.3 Automatická analýza dat

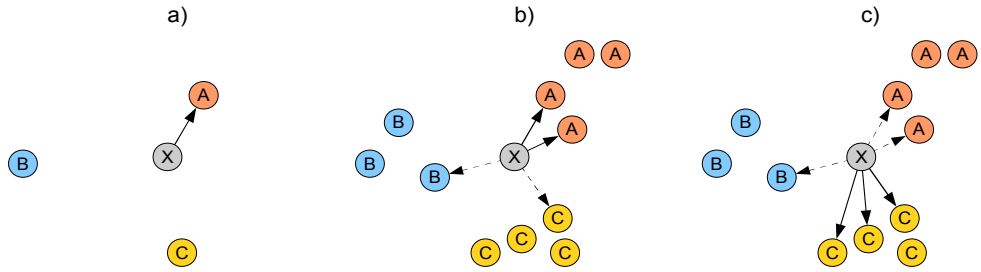
Pro automatické analyzování velkého objemu dat je často používán anglický termín „Data mining“. Jedná se ve své podstatě o vyhledávání zajímavých a v mnoha případech i skrytých informací v datech. Velice často je možné narazit na tento pojem v souvislosti s marketingem. Pro příklad mohu uvést vyhledávání potencionálních cílových zákazníků určité skupiny produktů. Nicméně je možné setkat se s tímto pojmem i v jiných odvětvích. V této diplomové práci je jedním z úkolů vyhledávání stejných či podobných úseků v naměřených datech. Nebude tedy nutné nasadit příliš sofistikované metody, ale měla by postačit metoda klasifikace podle nejbližší vzdálenosti od etalonu, případně metoda k-nejbližších sousedů. Uvedené metody jsou podrobněji rozepsány v [6] a [10] a já jsem těchto zdrojů využil pro sepsání kapitol 2.3.1 a 2.3.2.

2.3.1 Metoda klasifikace podle etalonu, k-nejbližších sousedů

Princip klasifikace podle vzdálenosti od etalonu je principiálně jednoduchý. Mějme několik etalonů popsaných určitým počtem parametrů. Z těchto parametrů je vytvořen vektor. Každý etalon tak reprezentuje jeden vektor v n-rozměrném prostoru, kde n udává počet parametrů etalonu. Rozpoznávání probíhá tím způsobem, že se pro

rozpoznávaný prvek vytvoří stejný vektor z parametrů jako u etalonů. Následuje určení vzdáleností od jednotlivých etalonů a přiřazení k tomu etalonu, kde je tato vzdálenost nejmenší. Kromě euklidovské vzdálenosti je možné využít i jiné druhy.

Metoda k-nejbližších sousedů rozšiřuje výše uvedenou metodu. Každý prvek, který má být rozpoznáván, není reprezentován jedním etalonem, ale více reprezentanty. Každý reprezentant je stejně jako etalon popsán vektorem parametrů. Při rozpoznávání se opět určí vzdálenosti neznámého prvku od reprezentantů. Poté se vybere definovaný počet nejmenších vzdáleností. Tento počet je značen parametrem k. Neznámý prvek je poté prohlášen za příslušníka té skupiny, k jejímž prvkům má nejvíce nejmenších vzdáleností. V případě kdy k = 1, jedná se ve své podstatě o metodu podle vzdálenosti od etalonu. Pro názornost jsem zařadil obrázek 2.4.



Obr. 2.4: Princip metody nejbližších sousedů
a) k = 1; b) k = 4; c) k = 6

Ve vyvíjené aplikaci však bude nutné uvedené algoritmy mírně modifikovat. Je to z toho důvodu, že se nejedná o přiřazení neznámého průběhu do určité skupiny, ale o rozhodnutí, zda jsou si dva signály navzájem podobné. V první řadě se nabízí modifikovat metodu klasifikace podle vzdálenosti od etalonu. Místo hledání nejmenší vzdálenosti se však bude vyhodnocovat jen vzdálenost mezi porovnávanými průběhy. Na základě velikosti této vzdálenosti je poté možné určit, zda jsou si signály podobné či nikoli. V případě, kdy pro vyhledávání bude k dispozici více sobě navzájem podobných předloh, může být výhodné použít modifikovanou metodu k-nejbližších sousedů. Modifikace spočívá ve vyhodnocení průměrné vzdálenosti porovnávaného úseku dat a předloh.

2.3.2 Metoda přímého vzájemného porovnání

V předchozích metodách byl signál popsán určitým počtem parametrů, které určovaly vektor v prostoru. Na základě vzdáleností se poté usuzovalo na podobnost

průběhů. V metodě přímého porovnávání se jako parametry použijí hodnoty spočtené na základě obou signálů. Parametry tak určují vektor v prostoru, jehož délka značí míru neshody mezi signály. Při aplikování tohoto algoritmu na dva shodné signály je tedy výsledkem vektor s nulovou délkou. Jako parametry míry shody mezi signály se nabízí možnost použít Pearsonův korelační koeficient, hodnotu signálové korelace a průměrnou kvadratickou odchylku.

Pearsonův korelační koeficient

Jedním z možných parametrů, jak určit podobnost dvou úseků dat, je Pearsonův korelační koeficient. Jeho hodnota se pohybuje v rozmezí od mínus jedné do jedné. Tato skutečnost je velice výhodná, neboť odpadá problém s různými efektivními hodnotami porovnávaných signálů. Porovnávané signály lze prohlásit za závislé pro hodnoty Pearsonova korelačního koeficientu blížícího se k jedné. Aby platila podmínka, že pro shodné signály má být velikost výsledného vektoru nulová, je nutné od výsledku odečíst jedničku. Po této úpravě dostaneme největší neshodu u navzájem nepřímo závislých signálů. Pro uvažované použití tohoto parametru tedy není odečtení hodnoty jedna na závadu.

Signálová korelace

Druhým parametrem, kterým je možné určit podobnost dvou průběhů, je signálová korelace. Standardní způsob výpočtu, kdy se dva signály přes sebe postupně posouvají a nejvyšší hodnota udává pozici největší shody, je však nutné mírně modifikovat. Je možné využít skutečnosti, že se porovnávají dva stejné časové úseky. Není tedy nutné signály přes sebe posouvat, ale je možné umístit je přímo přes sebe. Vzniká však problém s efektivní hodnotou signálů. Dva velice podobné signály, avšak s malou efektivní hodnotou, by byly vyhodnoceny jako nestejné, zato signály s velkou efektivní hodnotu, i když rozdílné, by byly vyhodnoceny jako shodné. Z tohoto důvodu je nutné před samotným výpočtem signálové korelace vydělit oba porovnávané signály svojí vlastní efektivní hodnotou. Poté již lze na základě výsledných hodnot rozhodnout o podobnosti signálů. Naskýtá se další problém, a to s určením délky vektoru pro vyhodnocení. Hodnota signálové korelace vypočtená popsaným způsobem musí zákonitě vycházet větší pro delší signály a naopak. Z tohoto důvodu je nutné výslednou hodnotu vydělit počtem vzorků v porovnávaných signálech. Touto drobnou úpravou je zajištěno, že pro libovolná data s různými efektivními hodnotami a počtem vzorků budou vycházet vždy podobné výsledky. Stejně jako v předešlém případě, i u hodnoty

signálové korelace, jsou si signály nejvíce podobné v okolí hodnoty jedna. Je tedy nutné před samotným vyhodnocením délky vektoru odečíst jako v předešlém případě od výsledku hodnotu jedna.

Průměrná kvadratická odchylka

Posledním parametrem, kterým je možné v porovnávacím algoritmu ohodnotit podobnost dvou signálů, je průměrná kvadratická odchylka. Ze stejného důvodu jako u metody výše, i průměrná kvadratická odchylka se vypočítává ze signálů, které jsou předtím vyděleny vlastní efektivní hodnotou. Vypočtenou hodnotu však není potřeba dále upravovat v závislosti na délce dat, protože dělení počtem vzorků je obsaženo již ve výpočtu samotné průměrné kvadratické odchylky. Na rozdíl od předešlých dvou způsobů, při použití průměrné kvadratické odchylky, jsou si signály tím více podobné, čím menší je vypočtená hodnota a pro naprosto totožné signály je tato hodnota rovna nule. Tato skutečnost přesně koresponduje s využitím vyhodnocení na základě délky vektoru, proto není nutné jako u předchozích dvou metod dělat žádné další úpravy.

3 Realizace programu

3.1 Modelování naměřených dat

Pracovat přímo s reálnými naměřenými hodnotami není vždy optimální. Příčinou může být jejich veliké množství nebo zašuměný průběh. Je proto snaha nahradit data určitým modelem. Je možné vytvořit jeden model pro všechna data, nebo více modelů rozdělených např. po měsících nebo týdnech. Model je vytvářen pomocí množiny bodů, které jsou následně proloženy křivkou určitého typu. Problematikou takového postupu je výběr správných bodů, pomocí kterých by se vytvořil co nejlepší model. Pro posuzování kvality modelu postačí obyčejné kvadratické kritérium.

3.1.1 Předzpracování reálných dat

Naměřená data bývají zpravidla zašuměná, či obsahují výrazné výkyvy, které nemají vypovídající charakter o celkovém průběhu měřené veličiny. Snahou je pomocí předzpracování dat tyto nedostatky odstranit či minimalizovat. Bez předzpracování dat by nebylo možné využít např. metody lokálních maxim a minim.

Běžným postupem je výpočet aritmetického průměru pro každý bod z jeho blízkého okolí. Velikost okolí, které je do průměru zahrnuto, je možné libovolně volit. Jediný problém nastává pro okrajové hodnoty, pro které není možné počítat průměr z neexistujících hodnot. U okrajových bodů je tedy nutné počítat průměr z menšího počtu okolních bodů. Ve svém průměrovacím algoritmu využívám pro okrajové body vždy maximální možnou velikost okolí, ze kterého je průměr pro daný bod počítán.

3.1.2 Model s rovnoměrně rozloženými interpolačními body

Nejjednodušší model lze získat tak, že se vyberou s rovnoměrným rozestupem určité body z naměřených, případně předzpracovaných dat a ty následně proložit zvolenou křivkou. Vytvoření modelu je velice jednoduché, ale odchylka od skutečných dat je ve většině případů příliš veliká. Jedinou možností, jak lépe interpolovat reálná data je použít více bodů pro tvorbu modelu. Model s rovnoměrně rozdělenými interpolačními body se v programu běžně nepoužívá, ale je využíván jako vstup do dalších optimalizačních algoritmů, které mají za úkol vybrat vhodnější body pro tvorbu modelu.

3.1.3 Model s postupnou optimalizací

Model s postupnou optimalizací vychází z modelu s rovnoměrně rozloženými interpolačními body. Samotná optimalizace spočívá v posunutí interpolačních bodů tak, aby se dosáhlo lepšího modelu. Posunují se všechny body, vyjma hraničních bodů. Posunování probíhá postupně, každý bod se tedy posunuje samostatně. Nejprve je nutné určit, kterým směrem se bude daný bod pohybovat. Vypočítají se proto dvě odchylky, pro posun o pozici doprava a pro posun o pozici doleva. V případě, kdy ani jeden model není lepší, pokračuje se optimalizací následujícího bodu a s dosavadním bodem se nehýbe. Pokud jsou obě odchylky menší než předchozí odchylka modelu, posunuje se bod tím směrem, kde je model lepší. Posunování probíhá do doby, než je odchylka modelu horší než byla v předchozím kroku. Druhou limitující podmínkou je pozice sousedního bodu. Posunování se zastaví i v okamžiku, že se bod posunul až do těsné blízkosti jiného bodu.

Tímto způsobem se provede optimalizace všech bodů. Celý proces posunování bodů se opakuje podle vstupního parametru funkce. Možnost opakování celého cyklu jsem zařadil z důvodu závislosti celého modelu na pozici všech bodů. Bylo by možné počet opakování určovat automaticky z podmínky, kdy se již neposunul žádný bod. Kvalita tohoto optimalizačního způsobu však není příliš veliká, proto jsem se touto možností nezabýval.

3.1.4 Model využívající lokální maxima a minima

Přirozenou úvahou je interpolovat libovolný průběh signálu propojením lokálních maxim a minim. Této myšlenky jsem využil u své další optimalizační metody. Algoritmus se však musel značně upravit, jelikož lokálních maxim a minim není ve většině případů přesně stejný počet, jaký je požadován pro interpolaci. Je tedy nutné některá lokální maxima či minima pro výsledný model nevyužívat, nebo naopak nějaké body pro tvorbu modelu přidat. Další neméně důležitou součástí této optimalizační metody je předzpracování vstupních dat. Jelikož nebývá vstupní signál příliš hladký, obsahoval by veliký počet lokálních maxim a minim a celý algoritmus by tak nemusel fungovat korektně. Po vyhlazení dat se předpokládá, že v průběhu zůstanou pouze korektní maxima a minima.

Pokud nastane případ, kdy je v již vyhlazeném průběhu více lokálních maxim a minim, je nutné některé tyto body vyřadit a pro model s nimi neuvažovat. Algoritmus odebírání těchto bodů pracuje na principu vzdálenosti maxim a minim na ose X. Počítá

se s tím, že blízké hodnoty maxim a minim lze shrnout do jednoho bodu a použít pro tvorbu modelu pouze tento jediný bod. Body se odebírají v cyklech, kde se v každém cyklu určí nejbližší lokální maximum a minimum a bod s menší hodnotou na ose X se odebere. Tento cyklus se opakuje do té doby, než zůstane potřebný počet bodů pro vytvoření modelu.

Druhou možností je, že je počet lokálních maxim a minim menší než potřebný počet bodů pro interpolaci. V tomto případě je nutné naopak některé body do modelu přidat. Opět se pracuje v cyklech, kde se v každém kroku určí největší vzdálenost mezi lokálním maximem a minimem. Do středu mezi tyto dva body se poté vloží další bod vstupující do tvorby modelu. Celý cyklus se opět provádí do té doby, než se získá potřebný počet bodů.

Výsledný průběh tohoto optimalizačního algoritmu na pohled věrněji kopíruje reálná data, nicméně je tento průběh na pohled od originálního průběhu posunutý a kvadratické kritérium dosahuje přibližně obdobných hodnot, jako při optimalizaci postupným posunováním bodů. V tomto místě se nabízí zkombinovat obě optimalizační metody dohromady, tedy začít s rozmístěním bodů do lokálních maxim a minim a následně provést posun jednotlivých bodů. Výsledky však ani v tomto případě nebyly výrazně lepší, proto jsem tento způsob nevyužil.

3.1.5 Vrchní a spodní model

V některých případech by bylo výhodné mít k dispozici „obálku“ signálu. Jedná se tedy opět o proložení dat určitým typem křivky, ale musí být splněna určitá pravidla. Pro vrchní hranici obálky musí platit, že všechny body naměřeného signálu musí ležet pod interpolovanou křivkou. U spodní hranice obálky je tomu přesně naopak.

Nasazení genetického algoritmu na tuto problematiku však není triviální záležitostí. Problém nastává u ohodnocení kvality interpolace. V případě, kdy by celý interpolovaný průběh splňoval výše uvedený požadavek, algoritmus by fungoval. Při porušení podmínek by však bylo nutné ohodnotit tuto interpolaci jako zcela nepřípustnou. Z důvodu těchto komplikací jsem aplikoval jiný typ optimalizace. Vytvoření vrchního a spodního modelu jsou dva navzájem téměř shodné algoritmy, proto popíšu princip výpočtu pouze spodní obálky. Vrchní se vytváří naprosto analogicky.

Ačkoli bylo uvedeno, že genetický algoritmus nelze použít přímo, přesto se v první části procesu výpočtu využije. Vytvoří se běžná optimalizovaná interpolace

průběhu. V další části se určí všechny odchylky interpolovaného a skutečného průběhu. Z těchto hodnot se vybere maximální hodnota. Je nutné uvést, že odchylku je nutné počítat jako rozdíl skutečného a interpolovaného průběhu. Pokud by se pořadí průběhů otočilo, bylo by nutné vybírat minimální hodnotu a v dalších výpočtech počítat s její absolutní hodnotou. Po určení největší odchylky se celý interpolovaný průběh posune o tuto hodnotu směrem dolů. Po této úpravě je již splněna podmínka, že všechna naměřená data leží nad interpolovaným průběhem. Nicméně výsledek by nebyl příliš uspokojivý.

Z tohoto důvodu je nutné provést další úpravy interpolovaného průběhu. Hlavní myšlenkou těchto úprav je postupné posouvání jednotlivých bodů modelu směrem nahoru. Při aplikaci tohoto postupu je však nutné si uvědomit skutečnost, že změnou jednoho bodu ovlivníme jeho celé okolí. Proto je nutné po každém posunutí bodu kontrolovat, zda nedošlo k porušení podmínek na obálku průběhu.

Posouvání bodů je možné dvěma různými způsoby. První možností je každý bod posouvat tak dlouho, než bude porušena podmínka pro tento druh interpolace. Posouvání se tedy v tomto případě provede tolikrát, kolik bodů vytváří modelovaný průběh. Výsledný interpolovaný průběh však není příliš kvalitní a vyskytují se v něm výrazné špičky. Proto bylo nutné posouvat body postupně.

Myšlenka postupného posouvání bodů je následující. Na začátek se zvolí určitá velikost kroku, o který se budou postupně jednotlivé body posouvat. Dojde-li k situaci, že nastane porušení podmínek na interpolovaný průběh, zmenší se na konci cyklu krok na polovinu a celý proces se opakuje. V tomto případě by se však algoritmus nikdy neukončil, proto je nutné konec podmínit určitou minimální velikostí kroku. Výsledný průběh také není bez výraznějších špiček, nicméně určení přesnější obálky signálu není potřeba, proto si myslím, že tento typ optimalizace postačuje.

3.1.6 Ukládání a načítání modelu

V úvodu jsem již naznačil možnost využití modelu pro zmenšení objemu dat potřebných k popsaní reálných hodnot. Bylo tedy nutné vymyslet způsob, jakým se data budou ukládat do souboru, aby bylo možné po jejich načtení s modelem v programu pracovat stejným způsobem, jako když se model vytváří přímo z dat z databáze. Z tohoto důvodu je nutné, mimo hodnot definujících model, uložit navíc několik čísel, pomocí kterých se správně dekódují načítaná data.

V prvé řadě se hodnoty modelu i čísla popisující tento model převedou na pole bajtů, které je již poměrně jednoduché zapsat do souboru. Vytvoření tohoto pole bajtů je ve své podstatě kódování, které je možné rozdělit na dvě části. V první části se seřadí všechny hodnoty zapisované do souboru za sebe. V druhé části se každá tato hodnota převede na bajty a zapíše do pole.

Strukturu pole je možné popsat následujícím způsobem. První hodnota udává počet bodů, ze kterých je model tvořen. Druhá hodnota značí počet modelovaných průběhů. Toto číslo musí být vždy sudé, jelikož každý průběh potřebuje ke svému popsání hodnoty na ose X a hodnoty na ose Y. V případě, že se ukládá jeden model, je tato hodnota tedy rovna dvěma. Další číslo je označení typu interpolace. Každému typu interpolace je přiřazeno jedinečné číslo, pomocí kterého je možné při načítání dat rozhodnout, jakým typem křivky budou body proloženy. Všechny uvedené hodnoty jsou celočíselné proměnné, takže celá hlavička souboru zabírá 12 bajtů. Poté následují hodnoty samotného modelu. Jejich počet je dán prvními dvěma hodnotami v hlavičce. Tyto hodnoty jsou však neceločíselné a každá hodnota tak zabírá 8 bajtů. Poslední místa v poli jsou určena pro hodnoty odchylek jednotlivých modelů od reálných dat. Po načtení modelu ze souboru již není možné přistupovat k originálním datům a odchylku modelu dopočítávat. Těchto odchylek je stejný počet jako ukládaných modelů a jedná se opět o neceločíselné hodnoty.

Tímto způsobem seřazené hodnoty se ve druhé části převedou na pole bajtů, které se uloží. Při následném načítání uložených hodnot je nejprve nutné určit o jaký typ interpolace se jedná. Následuje dekódování dat z pole bajtů a proložení jednotlivých modelů správnou křivkou. Nakonec se dekódují odchylky pro jednotlivé modely.

3.2 Genetický algoritmus

Výše uvedené optimalizační metody nejsou založené na žádném matematickém principu optimalizace. Jsou založené především na úvahách, jakým způsobem by volil interpolační body sám člověk. Výsledky těchto metod, až na výjimečné případy, nedosahují požadovaných kvalit. Proto bylo nutné zvolit pro optimalizaci vhodnou a ověřenou metodu. Složitost řešené úlohy vedla k volbě genetického algoritmu.

Genetický algoritmus pracuje s populací jedinců. Každý jedinec je tvořen určitým počtem genů, které ho jednoznačně charakterizují. Cílem genetického algoritmu je postupným vývojem generací dospět do takového bodu, kdy poslední generace obsahuje alespoň jednoho jedince, který nejvíce odpovídá optimálnímu řešení. Jeden stupeň

vývoje nové generace je založen na provedení jedné ze dvou optimalizačních metod. Jedná se o křížení či mutování původních jedinců. Ve svém programu jsem vytvořil čtyři různé způsoby křížení a mutování.

3.2.1 Omezující podmínky genetického algoritmu

Genetický algoritmus často pracuje s dvojkovou reprezentací genů v jedincích. Ve vyvíjené optimalizaci jsou však geny tvořeny celými nezápornými čísly. Další odlišností je skutečnost, že jedinec nesmí obsahovat dva stejné geny, což je dáno použitím interpolačních funkcí z knihovny Math.NET, které by při nesplnění tohoto požadavku nepracovaly správně či vůbec. Poslední atypickou podmínkou každého jedince v populaci je nutnost existence dvou speciálních genů. Jedná se o gen s nulovou hodnotou a o gen s hodnotou maximálního indexu originálních dat. Při nedodržení této podmínky by se interpolace provedla, ale ne v celém rozsahu originálních dat.

Dále jsem použil pravidlo, díky kterému je počet jedinců v populaci vždy dělitelný číslem čtyři. Tato hodnota má své opodstatnění, které spočívá ve výběru poloviny dobrých a poloviny špatných jedinců v generaci a jejich následném křížení a mutování. Kdyby počet jedinců v generaci nesplňoval požadavek na dělitelnost čtyřmi, mohl by nastat problém při rozdělování na dobré a špatné jedince či vzniku nových potomků.

3.2.2 Společná část genetického algoritmu

Ačkoli moje aplikace obsahuje čtyři různé metody genetické optimalizace, některé části jsou u všech metod stejné. Jedná se o vytváření počáteční populace, výpočet ohodnocení jednotlivých jedinců a testování konce celého genetického algoritmu. Dále jsou stejné postupy při výběru dobrých a špatných jedinců.

Vytváření počáteční populace

Aby bylo možné provést genetickou optimalizaci, je nutné na začátku náhodně vygenerovat počáteční populaci. Všichni jedinci v této nulté populaci jsou tedy vygenerováni zcela náhodně, avšak musí být dodrženy omezující podmínky uvedené výše. Vstupními parametry funkce generující počáteční populaci jsou hodnoty udávající počet jedinců v populaci, počet genů v jedinci, originální data rozdělená samostatně na hodnoty X a Y. Posledním parametrem je typ interpolace, která se bude během optimalizace využívat.

Generování populace začíná překontrolováním podmínky, zda zadaný počet jedinců splňuje dělitelnost čtyřmi. Pokud ne, je počet jedinců v populaci navýšen tak, aby podmínku splňoval. Následuje vytvoření zadaného počtu prázdných jedinců, kteří neobsahují žádný gen. Až když jsou vytvořeni všichni jedinci, začne se s přidáváním náhodně vygenerovaných genů do každého jedince. Během generování se průběžně kontroluje, zda jedinec neobsahuje gen stejné hodnoty jako je vygenerovaná hodnota. V tomto případě se generování opakuje. Protože musí být i v počáteční populaci splněna podmínka, že každý jedinec obsahuje gen s hodnotou nula a maximální hodnotou indexu originálních dat, jsou geny s uvedenými hodnotami do každého jedince přidány.

Po vygenerování všech jedinců je nutné k nim přiřadit ohodnocení. K ohodnocení je použito kvadratické kritérium mezi interpolovaným a skutečným průběhem. Funkce na výpočet této hodnoty je popsána níže.

Výpočet ohodnocení jedince

Jak bylo řečeno v předešlém odstavci, k ohodnocení každého jedince je využito kvadratického kritéria. Kvadratické kritérium udává rozdíl mezi interpolovaným a původním průběhem, přičemž rozdíl mezi těmito dvěma hodnotami se počítá jako kvadrát. Vstupem do této funkce je jeden jedinec, originální průběh rozdělený na hodnoty X a Y a typ interpolace.

Výpočet kvadratické odchylky začíná vyseparováním hodnot z originálních dat podle zadaného jedince. Geny v jedinci přesně odpovídají indexům v originálních datech, což tuto separaci značně usnadňuje. Následuje vytvoření modelu z vyseparovaných hodnot. Pro určení typu křivky slouží vstupní parametr, který tuto křivku jasně určuje. Po vytvoření interpolace se určí kvadratická odchylka modelu od původních dat. Tato hodnota se následně vrací jako výsledek ohodnocovací funkce.

Testování ukončení genetického algoritmu

Genetický algoritmus je možné provádět v předem přesně stanoveném počtu kroků. Z důvodu efektivnosti je však lepší optimalizaci ukončit v případě, kdy po křížení nebo mutaci dostáváme stále stejné výsledky. Rozhodujícím parametrem je proto velikost odchylky nejlepšího jedince. Vstupem do této funkce je pouze seznam s vývojem těchto odchylek a hodnota udávající počet po sobě jdoucích stejných hodnot, při kterých se algoritmus ukončí.

Na tomto místě by bylo vhodné uvést skutečnost, že seznam s vývojem minimální odchylky není uložen přímo v generaci, ale je nutné jej vytvářet během samotné

optimalizace externě. Stejné je to s ukončením optimalizačního algoritmu. Tato funkce optimalizaci přímo nezastavuje, pouze vrací logickou hodnotu pravda, nepravda, na jejímž základě je možné rozhodnut o ukončení genetického algoritmu.

Výběr dobrých a špatných jedinců

Základem genetického algoritmu je křížení a mutace jedinců. Vznik lepšího potomka je pravděpodobnější, pokud se kříží nebo mutují dobří jedinci. Každý jedinec je ohodnocen hodnotou, která reprezentuje jeho zdatnost. V tomto případě se jedná o hodnotu určující odchylku interpolovaného a originálního průběhu. Čím je tato odchylka menší, tím zdatnější a lepší je jedinec. Ani křížení a mutování nejlepších jedinců však nemusí zaručit vznik nových a lepších potomků. Z tohoto důvodu je vhodné nejlepší jedince přemístit do nové generace nepozměněné.

V mém genetickém algoritmu zanechávám přesně polovinu lepších jedinců. Horší polovina jedinců je smazána a následně nahrazena potomky, kteří vzniknou křížením a mutací ze zachovaných jedinců. Rozdělení populace na dobrou a špatnou část není nikterak obtížné, když je každý jedinec ohodnocen. Důležitou součástí rozdělení populace je však vytvoření dvou polí, ve kterých jsou uvedeny indexy dobrých a špatných jedinců v populaci. Vzhledem k dalšímu použití je třeba mít pole s indexy dobrých jedinců seřazené postupně od nejlepšího jedince. Odůvodnění této skutečnosti je dále v textu.

3.2.3 Jednobodové křížení a mutace

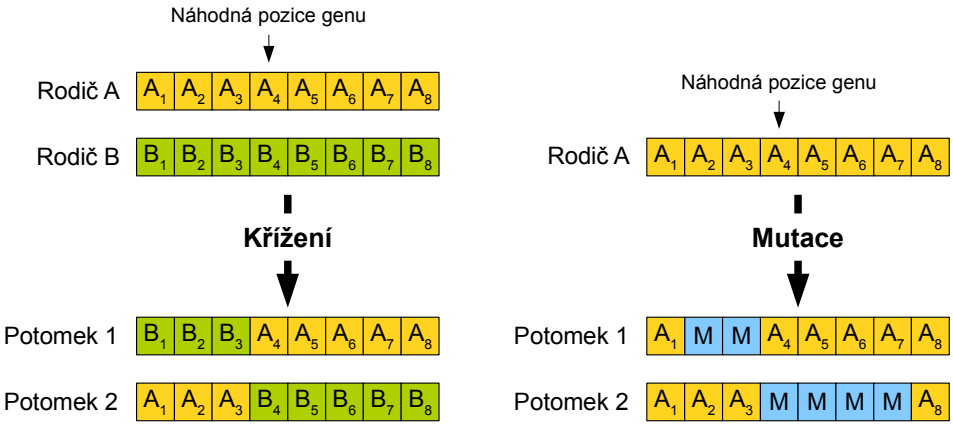
Jednobodové křížení a mutace představuje nejjednodušší způsob vytváření potomků. V principu se jedná o vygenerování náhodného čísla, které udává pozici genu v rodičích, od kterého se zbylé geny překříží či zmutují.

Důležitý je také výběr dvou rodičů pro křížení, nebo jednoho rodiče pro mutaci. Při křížení se jeden rodič bere postupně ze seznamu dobrých jedinců, druhý je k němu vygenerován náhodně. K mutaci stačí jen jeden rodič a ten se vybírá opět postupně ze seznamu dobrých jedinců. Po křížení i mutaci vznikají vždy dva noví jedinci. Tím je zaručeno, že nejlepší čtvrtina jedinců z celé populace bude vždy křížena či mutována.

Ke křížení, jak bylo uvedeno výše, jsou vybráni dva zdatní jedinci a dále je vygenerována náhodná pozice jednoho genu. Dva noví jedinci vzniknou překřížením rodičů kolem vybraného genu, což je patrné na obrázku 3.1. Při křížení však nesmí být porušena žádná z omezujících podmínek na jedince. Z tohoto důvodu je nutné při křížení genů vždy kontrolovat, jestliže se stejný gen v jedinci již nevyskytuje. Pokud

tento případ nastane, gen z rodiče se nepoužije, ale je náhodně vygenerován jiný. Výslední potomci se uloží do generace na pozice uvedené v seznamu špatných jedinců.

Při mutování je využit pouze jeden rodič, ale vznikají dva noví jedinci. Opět je náhodně vybrána pozice jednoho genu, která určuje místo mutace. První potomek vznikne mutováním všech genů před vygenerovanou pozicí a zachováním genů za touto pozicí. Princip mutace je patrný na obrázku 3.1. Druhý potomek vzniká přesně opačně. Opět je důležité neustále dodržovat omezující podmínky na jedince. Stejně jako v případě křížení při shodě dvou genů je druhý gen nahrazen náhodnou hodnotou. Potomci se opět ukládají na pozice špatných jedinců.



Obr. 3.1: Ukázka principu jednobodového křížení a mutace

Postupným ukládáním potomků na místa špatných jedinců se docílí opětovného zaplnění celé populace. Z principu křížení a mutace je zároveň patrná podmínka na velikost populace dělitelná čtyřmi.

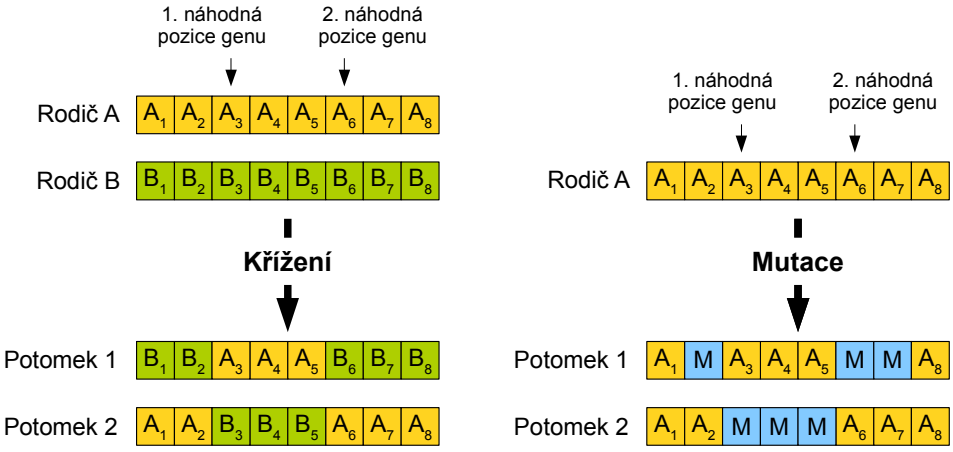
3.2.4 Dvoubodové křížení a mutace

Dvoubodové křížení a mutace je rozšířením předešlého způsobu vytváření potomků. Základem je vygenerování dvou náhodných hodnot, které určují pozice genů v rodičích. Křížení a mutace probíhá podobně jako v předešlém případě s tím rozdílem, že body, kolem kterých křížení probíhá, jsou dva. Výběr dvou rodičů pro křížení a jednoho rodiče pro mutaci je naprosto totožný s jednobodovým způsobem.

Vstupem křížení jsou dva jedinci vybraní způsobem popsaným v jednobodovém křížení. Dále jsou náhodným způsobem vygenerovány pozice dvou genů. V případě, že by obě vygenerované pozice byly shodné, jednalo by se o jednobodové křížení, proto se tato skutečnost kontroluje a v případě stejných vygenerovaných hodnotách se náhodný výběr jedné hodnoty opakuje. Dva noví jedinci vzniknou překřížením rodičů kolem

vybraných genů. Způsob křížení je zobrazen na obrázku 3.2. Pro dodržení omezujících podmínek jedince je nutné při kolizi dvou stejných hodnot v genech provést mutaci příslušného genu. Výslední dva potomci se uloží v generaci na pozice špatných jedinců.

I v případě dvoubodové mutace je využit pouze jeden rodič, ale vznikají dva noví jedinci. Náhodně jsou vybrány pozice dvou genů, které určují místo mutace. Pro generování pozic genů platí stejné pravidlo jako při křížení. To znamená, že se pozice nesmí shodovat. První potomek vznikne mutováním všech genů před vygenerovanou pozicí prvního, náhodně vybraného genu. Mezi vybranými geny se hodnoty v jedinci zachovávají a za druhým náhodně vybraným genem opět probíhá mutace. Princip mutace je patrný na obrázku 3.2. Druhý potomek vzniká přesně opačně. I zde se neustále kontroluje dodržení omezujících podmínek jedince. Výslední potomci se ukládají na pozice špatných jedinců.



Obr. 3.2: Ukázka principu dvoubodového křížení a mutace

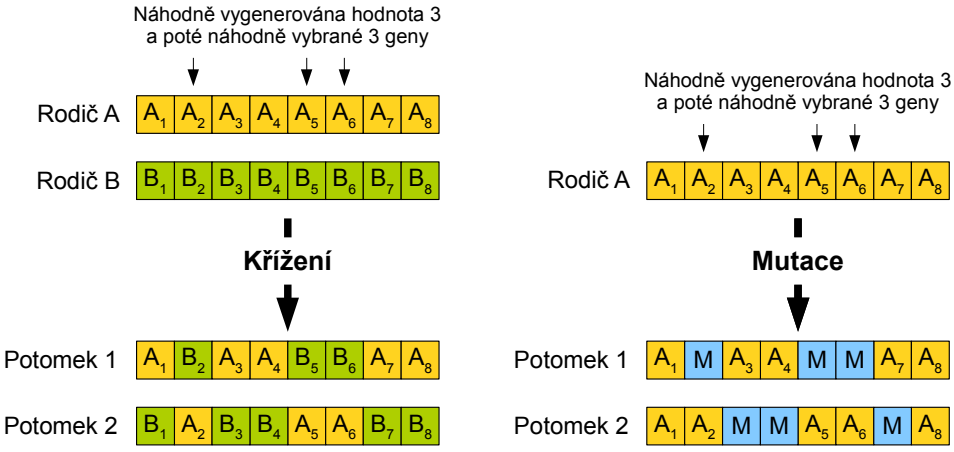
3.2.5 Náhodné křížení a mutace

Jednobodové a dvoubodové křížení a mutace jsou standardními způsoby, které se v genetické optimalizaci používají. Náhodné křížení a mutaci jsem odvodil pomocí předešlých metod svým vlastním způsobem. Hlavní myšlenkou je křížit a mutovat náhodný počet genů na náhodných pozicích. Výběr genů, které se mají křížit a mutovat, je od předešlých dvou způsobů značně odlišný. Naopak výběr rodičů je naprosto totožný s předešlými metodami.

Pro křížení jsou zapotřebí standardně dva jedinci. Následuje nejdůležitější část metody náhodného křížení a mutace. V první fázi se vygeneruje jedno číslo udávající počet genů, které se budou křížit. Generuje se hodnota v rozmezí jeden až všechny

geny. Ve druhé fázi se vygenerují náhodné pozice příslušného počtu genů, který je určen číslem z první fáze. Při generování pozic genů se průběžně kontroluje, zda se některá pozice neopakuje. V takovém případě se musí hodnota generovat znova. Po kompletaci celého seznamu náhodných genů se přechází k vlastnímu křížení. Postupně se procházejí všechny geny a v případě, kdy se pozice genu vyskytuje ve vygenerovaných hodnotách, provede se křížení. V opačném případě se hodnoty genů zachovávají. Celý proces křížení je patrný na obrázku 3.3. Během celého algoritmu křížení se průběžně kontroluje, zda jsou splněny podmínky jedince. V případě konfliktu příslušný gen mutuje. Dva výslední jedinci nahrazují špatné jedince v generaci.

Pro mutování postačí jeden rodič. Generování genů, které budou mutovat, probíhá stejným způsobem jako u křížení. Vlastní mutace probíhá také téměř shodně. Zůstává zachováno postupné procházení genů a rozhodování o mutaci na základě výskytu pozice genu ve vygenerovaných hodnotách. Jelikož vznikají dva potomci, mutuje gen vždy jen v jednom potomkovi. Způsob mutování je znázorněn na obrázku 3.3. Potomci se opět ukládají na pozice špatných jedinců.



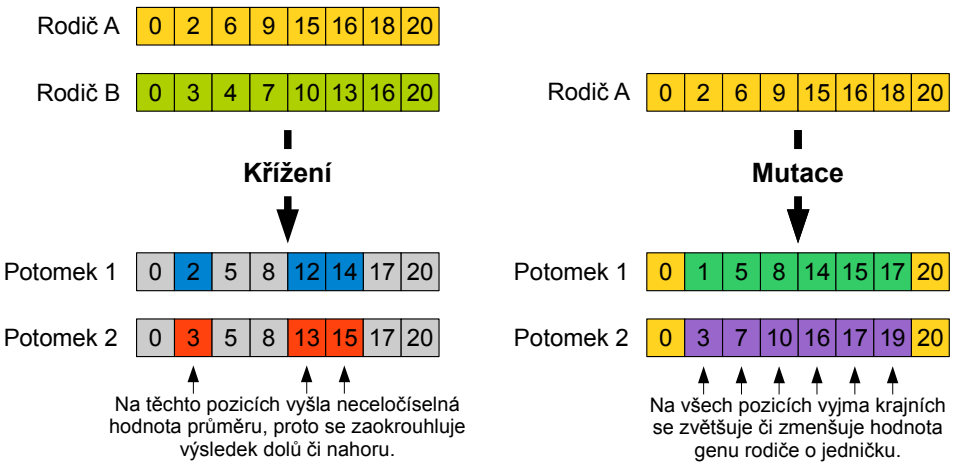
Obr. 3.3: Ukázka principu křížení a mutace s náhodným výběrem genů

3.2.6 Průměrované křížení a mutace

Tato metoda vznikla vzhledem ke skutečnosti, že hodnoty genů nejsou pouze binární jedna a nula, ale jsou z oboru celých nezáporných čísel. Naskytla se tak možnost využít pro křížení aritmetického průměru mezi dvěma geny z rodičů. V tomto případě se využijí při křížení i mutace vždy všechny geny v jedinci. Výběr rodičů zůstává stejných jako ve všech ostatních metodách.

Pro křížení se tedy používají dva rodiče a vznikají dva potomci. Jak bylo naznačeno, k výpočtu kříženého genu se používá aritmetický průměr. Z této skutečnosti se může zdát, že vznikají dva noví totožní jedinci. Díky práci s celými nezápornými čísly tomu tak však není. Po zprůměrování je nutné výslednou hodnotu genu zaokrouhlit na celou část. Právě díky zaokrouhlení vznikají dva odlišní jedinci. Využívá se toho, že do prvního jedince se ukládá hodnota genu zaokrouhlená směrem dolů a do druhého potomka hodnota genu zaokrouhlená směrem nahoru. V případě, kdy po průměrování vyjde celé číslo, uloží se do obou potomků stejná hodnota. Princip křížení je znázorněn na obrázku 3.4. Nutností je opět průběžné kontrolování podmínek jedince a případná mutace konfliktního genu. Výslední potomci se ukládají jako v předešlých metodách na pozice špatných jedinců.

Pro mutování se používá jeden rodič stejně jako v ostatních případech. Mutování se však provádí značně odlišným způsobem. Jak bylo řečeno výše, mutují všechny geny, kromě prvního a posledního, které musí splňovat příslušné podmínky. Mutování genů probíhá přičtením či odečtením jedničky od hodnoty genu rodiče. Vznikají tak dva potomci, kteří jsou od rodiče ve své podstatě posunuty o jedničku doprava či doleva. Princip mutace je znázorněn na obrázku 3.4. I v případě, kdy se přičítá a odečítá jednička od příslušného genu, může dojít ke shodné hodnotě dvou genů. Z tohoto důvodu je nutné tuto skutečnost průběžně kontrolovat a pro takový gen vygenerovat zcela novou náhodnou hodnotu. Noví potomci se opět ukládají na pozice špatných jedinců.



Obr. 3.4: Ukázka principu křížení a mutace s využitím průměrovací metody

3.2.7 Třída generace

Třída generace v programu slouží k definování jedné generace jedinců. Obsahuje několik datových typů, které jsou navzájem provázané a jsou významné, jak pro charakteristiku generace, tak pro funkce optimalizující tuto generaci.

Největší datovou strukturou je pole listů obsahující celočíselné proměnné. Celé pole listů tvoří populaci, kde každý jeden list odpovídá jednomu jedinci. Jedince definují geny, kterým odpovídají jednotlivé hodnoty v listu. Tyto celočíselné hodnoty jsou indexy z pole hodnot času a dat, které budou popsány dále. Využití celočíselných indexů oproti přímým hodnotám, které jsou reálná čísla, je úspora paměti a urychlení práce optimalizačního algoritmu.

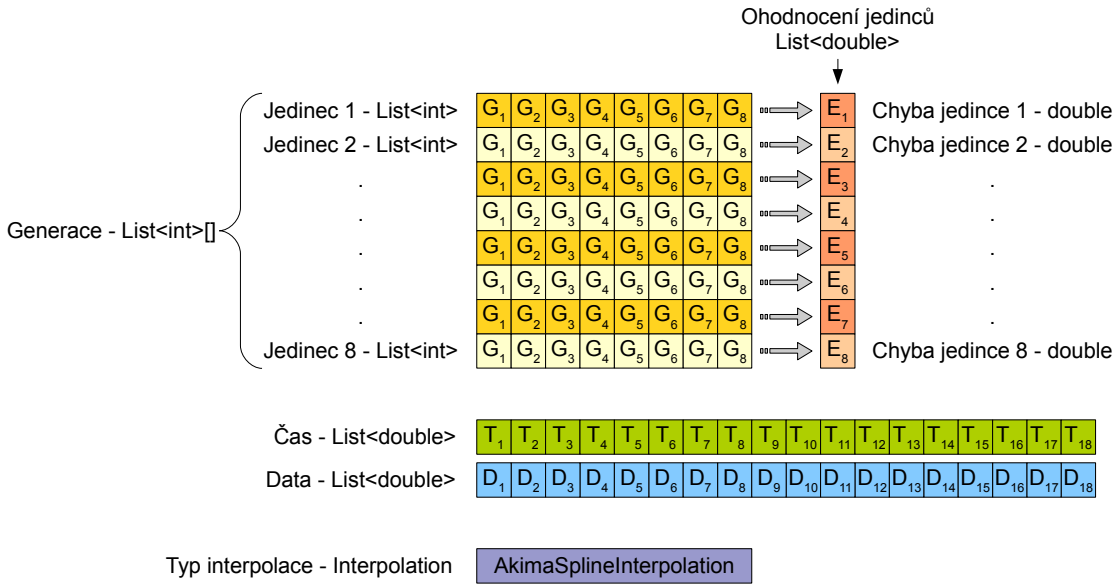
Každý jedinec v populaci musí být ohodnocen parametrem, který definuje jeho kvalitu. Pro určení kvality jedince je použito kvadratické kritérium. Každému jedinci tak náleží jedna hodnota z oboru reálných čísel, pro jejichž uložení je použit jeden list. Každá hodnota v listu uchovává chybu jednoho jedince.

Pro možnost optimalizování populace je nutné mít v generaci uložený průběh originálních dat. Ten je rozdělen na dvě samostatné části, a to na hodnoty času na ose X a na hodnoty dat na ose Y. K uložení dat jsou opět použity listy neceločíselných hodnot. Spolu s těmito daty souvisí další hodnota uložená v generaci. Jedná se o maximální hodnotu genu v jedinci. Ta je daná počtem hodnot v originálních datech a rovná se jejich počtu. Nesmí totiž nastat případ, aby gen v jedinci obsahoval index mimo rozsah originálních dat.

Poslední hodnotou uloženou v generaci je typ interpolace, podle které je počítána zdatnost jednotlivých jedinců. Dále se typ interpolace předává do optimalizačního procesu, kde se opět používá při ohodnocování jednotlivých jedinců.

Dále generace obsahuje několik funkcí, které vracejí specifické části generace. Tyto funkce slouží pro usnadnění práce s generací. Jedná se o funkce na výpočet minimální, maximální a průměrné chyby jedinců. Dále pak pro výběr nejlepšího jedince v populaci a určení jeho indexu.

Pro názornou ilustraci na obrázku 3.5 jsem použil generaci o osmi jedincích, přičemž každý jedinec má osm genů. Ke každému jedinci je přiřazena jedna hodnota udávající jeho zdatnost. Dále generace obsahuje originální data rozdělená na dvě samostatné části. V každé části je uloženo osmnáct hodnot. Posledním prvkem je typ interpolace. Ve vyobrazeném případě se jedná o AkimaSplineInterpolation.



Obr. 3.5: Grafické znázornění prvků ve třídě generace

3.3 Vyhledávání podobných datových úseků

Algoritmy programované během mé diplomové práce jsou určeny převážně pro práci s rozsáhlými daty. Ty obsahují záznamy o naměřených veličinách za celé dny, týdny až měsíce. Prohledávání takto rozáhlých dat jsem proto rozdělil na dva různé způsoby. První možností je vzít data za určitý úsek, např. za týden, rozdělit je na sedm částí, které reprezentují jednotlivé dny. Během vyhledávání následně porovnávat každý den s každým a hledat vzájemnou shodu či podobnost. Druhou možnou aplikací je vyhledávání určitého typického úseku v celých datech. Může se jednat např. o zjištění počtu výskytů atypického naměřeného průběhu během dlouhého časového horizontu.

3.3.1 Vzájemné porovnávání úseků dat

Jak bylo naznačeno výše, jedná se o porovnávání stejných úseků dat. K ohodnocení shody je použito třech parametrů, Pearsonova korelačního koeficientu, signálové korelace a průměrné kvadratické odchylky. V teoretické části bylo uvedeno, že se shoda určuje na základě výsledné velikosti vektoru, který definují zmíněné tři parametry. Aby bylo dosaženo stejného měřítka na všech osách, bylo nejdříve nutné získat hodnoty pro přepočet mezi jednotlivými parametry. Jelikož průměrná kvadratická odchylka má přesně tvar, který je pro vyhodnocování potřeba, zbylé dva parametry se přepočítávají právě k této hodnotě.

K získání potřebných dvou parametrů jsem využil 1000 náhodně vygenerovaných průběhů. Dalším krokem bylo vypočtení všech tří parametrů pro všechny kombinace průběhů. Těchto kombinací je celkem 499 500. Poté se pro každou dvojici průběhů vypočítal poměr mezi Pearsonovým koeficientem, či signálovou korelací vůči průměrné kvadratické odchylce. Ze všech hodnot se nakonec vypočítal průměr. Pro kontrolu jsem tento výpočet nechal proběhnout několikrát, pokaždé však vyšly téměř shodné hodnoty. Domnívám se tedy, že je možné je nadále používat. Konkrétní hodnoty pro přepočet jsou 2,425 pro Pearsonův koeficient a 0,2 pro signálovou korelaci.

Porovnávání s využitím všech parametrů

Tato metoda využívá všech tří parametrů k rozhodnutí o podobnosti či nepodobnosti porovnávaných dat. Jak bylo uvedeno výše, je potřeba nejdříve přepočítat všechny parametry do stejného měřítka. Po přepočtu tvoří tyto hodnoty vektor v trojrozměrném prostoru a jeho velikost ukazuje na míru shody mezi porovnávanými signály. Aby byla funkce využívající tohoto postupu komplexnější, zařadil jsem možnost vážení jednotlivých parametrů. Váhy mohou nabývat hodnot v rozsahu od nuly do jedné. Je tedy možné libovolně určovat závislost výsledků na jednotlivých parametrech. Dále jsem zařadil možnost zvolit si prahovou hodnotu délky výsledného vektoru, do které se porovnávané průběhy vyhodnocují jako shodné.

Porovnávání s využitím samostatných parametrů

Z výše uvedeného odstavce by mohlo vyplynout, že porovnávací funkce, využívající jednotlivých parametrů samostatně, jsou zbytečné. Nabízí se totiž možnost nastavit u předešlé funkce váhy zbývajících dvou parametrů jako nulové. Tato myšlenka však není zcela správná. Funkce využívající všech tří parametrů využívá tyto hodnoty až po jejich přepočtu, tedy s určitou chybou. Proto jsem vytvořil navíc tři samostatné funkce, které vyhodnocují shodu vždy jen v závislosti na jednom ze tří parametrů. V tomto případě se však jedná o vektor pouze v jednorozměrném prostoru. Funkce umožňují nastavit pouze prahovou hodnotu, protože váha parametru by v tomto případě byla nadbytečná.

3.3.2 Průběžné vyhledávání úseků dat

Při průběžném vyhledávání je hlavním cílem v rozsáhlých datech najít, spočítat a určit pozice předem zadaných kratších časových posloupností. Jako příklad je možné uvést vyhledávání poruch, které mají vždy podobný průběh. K samotnému

rozpoznávání je možné použít metody uvedené výše. Je-li k dispozici více předloh jednoho hledaného úseku, lze využít také modifikovanou metodu k-nejbližších sousedů. Nejkomplikovanější části celého algoritmu je určení reálných výsledků ze všech nalezených shod.

Algoritmus s využitím porovnávání pomocí všech parametrů

Princip činnosti tohoto vyhledávacího algoritmu je možné rozdělit na několik částí. Nejdříve se vyhodnocují parametry, které budou použity dále v algoritmu. Jedná se o počet předloh, které do vyhledávací funkce vstupují. Dalším důležitým parametrem je počet vzorků v předlohách. Jako poslední se určuje, kolik celkových porovnání se v algoritmu provede. Tato hodnota se vypočítává z počtu vzorků v datech, ve kterých se vyhledávání provádí, a na již zmíněném počtů vzorků v předloze.

Následuje postupné posouvání vzorků po prohledávaném signálu. Protože je možné zadat do funkce více předloh, musí se v každém kroku vypočítat hodnota délky vektoru pro každou dvojici úseku dat a předlohy. Jako finální výsledek předloh a daného úseku dat se dále v algoritmu počítá s průměrem z těchto hodnot. K samotnému výpočtu je použita funkce na porovnávání úseků s využitím všech parametrů. Aby algoritmus postupného vyhledávání z této funkce získával správná data, je nutné dodržet dvě podmínky. Vstupem musí být pouze dva signály, úsek dat a jedna předloha. Druhou podmínkou je nastavení velkého prahu, aby funkce vrátila hodnotu vždy. Možnost nastavování vah jsem pro postupné vyhledávání jako volitelné parametry nezařadil, ale nastavil je rovnoměrně pro všechny hodnoty rovné jedné.

Po zjištění průměrné délky vektoru dochází k vyhodnocení, zda je právě porovnávaný úsek dat podobný předlohám. Rozhoduje se na základě volitelného prahu. V případě, kdy je výsledná průměrná délka vektoru menší nebo rovna prahu, zařadí se index počátku úseku a hodnota délky vektoru do seznamu. Zde však nastává největší úskalí celého algoritmu. Z podstaty porovnávání je patrné, že pro předlohu posunutou o krátkou vzdálenost doleva či doprava od hledaného úseku, také dostaneme malé hodnoty délky vektoru. Aby nedocházelo k vícenásobnému označení jednoho místa, je nutné ze seznamu všech identifikovaných úseků vybrat pouze ty relevantní. Jelikož se jedná o rozsáhlejší problém a stejné řešení je nutné aplikovat i v následujícím algoritmu, vyčlenil jsem této problematice samostatnou podkapitolu.

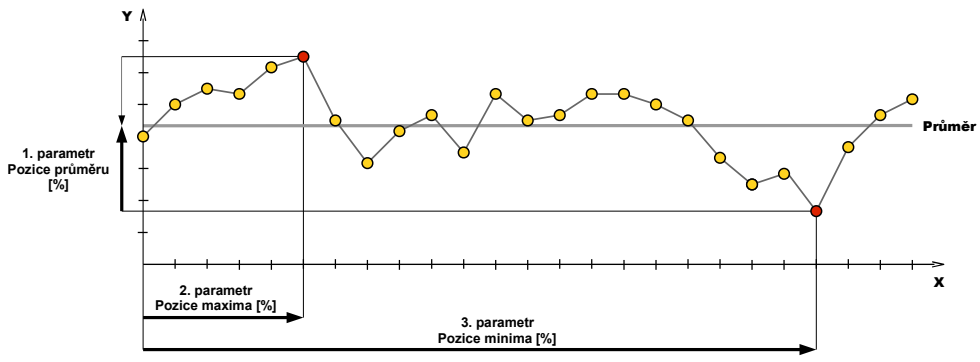
Algoritmus založený na metodě k-nejbližších sousedů

Postup výpočtu je obdobný jako u předchozí metody. Liší se však způsob, jakým jsou předlohy s částí dat porovnávány. Každý vzorek, stejně tak i vybraná část dat, jsou popsány určitými významnými parametry. Tyto parametry tvoří vektor v prostoru. Na základě porovnání vzdálenosti mezi těmito vektory lze vyhodnotit podobnost signálů. Aby metoda vykazovala dobré výsledky, je nutné zvolit správné parametry, kterými bude signál popsán.

Prvotní myšlenkou bylo využít statistických hodnot, jako je průměr, rozptyl a směrodatná odchylka. Naskýtá se však problém, jak eliminovat vliv efektivní hodnoty. Dalším problémem by se stalo určování prahu. Jako řešení se nabízí přeškálování signálu a výsledných hodnot do potřebného měřítka. Nicméně již při vývoji tohoto algoritmu bylo patrné, že tato kombinace hodnot charakterizujících signál, není výhodná. V porovnání s předešlým algoritmem dávala tato metoda odlišné výsledky a zjevně shodné úseky v některých případech vůbec neoznačila. Proto bylo nutné popsat signál jinými parametry.

Z důvodu odstranění vlivu efektivní hodnoty a možnosti nastavení jednotného prahu pro různé druhy signálů jsem zvolil popis signálu hodnotami, které lze vyjádřit procentuálně. Vzhledem ke skutečnosti, že navrhovaná metoda má sloužit k identifikaci specifických událostí, jsem jako charakteristické parametry signálu vybral pozici jeho průměru, maxima a minima. Celkem jsou tedy k dispozici tři parametry. Pozice průměru je uvažována v rozsahu mezi maximem a minimem, přičemž hodnota 0% odpovídá průměru rovnému minimu. Pozice minima a maxima se vztahuje k počátku popisovaného úseku na ose X. Pro názornost jsem zařadil obrázek 3.6.

Jako u předešlého způsobu vyhledávání, i u této metody dochází k vícenásobnému označení stejného místa. Proto je nutné opět nakonec algoritmu zařadit filtrování výsledků.



Obr. 3.6: Využívané parametry pro charakterizování signálu

Filtrování výsledků v průběžném vyhledávání

Obě metody narážejí na problém vícenásobného označení stejného hledaného úseku. Z tohoto důvodu je nutné na konci obou algoritmů provést selekci pouze těch hodnot, které odpovídají reálným výsledkům. Nejkritičtější částí je rozčlenění výsledků vyhledávání do skupin, které patří k jednomu stejnému místu. Využívá se dvou předpokladů. Prvním z nich je skutečnost, že chybné výsledky se nacházejí v těsném okolí výsledku správného. Druhým předpokladem je určitá posloupnost hodnot v tomto úseku. Algoritmus rozčlenění na jednotlivé úseky tedy prochází všechny výsledky a kontroluje, zda následující hodnota spadá do blízkého okolí. Pokud ano, jedná se stále o stejný úsek. V opačném případě je současná hodnota prohlášena za konec stávajícího úseku a následující hodnota za začátek úseku nového. Samostatně jsou ošetřeny počáteční a koncová hodnota v seznamu výsledků.

Po rozčlení na jednotlivé úseky je nutné vybrat ten výsledek, který dosáhl nejlepší hodnoty během porovnávání. Vybrané hodnoty jsou výstupem postupných vyhledávacích algoritmů. Za zmínku stojí dva speciální případy, kdy se popsané filtrování nemusí provádět. Jedná se o situaci, kdy je výsledkem postupného porovnávání jedna nebo žádná hodnota.

3.4 Zkušební aplikace

Z důvodu možnosti otestování naprogramovaných algoritmů jsem vytvořil klientskou aplikaci s grafickým rozhraním. Souhrn činností tohoto programu je možné rozdělit na tři hlavní části. V prvé řadě jde o aplikování dříve naprogramovaných algoritmů na reálná data. Jelikož mají tyto funkce volitelné vstupní parametry, je druhým účelem programu umožnit tyto parametry nastavovat. Posledním požadavkem na testovací program je přehledné zobrazení výsledků.

Z důvodu, že v mé diplomové práci nebylo účelem řešit problematiku načítání dat z databáze, využívá testovací aplikace jiný program, který tuto problematiku řeší. Až po načtení dat pomocí zmíněného programu se spustí moje testovací aplikace spolu s již načtenými daty.

Během své diplomové práce jsem se zaměřil na tři typy algoritmů. První z nich je zaměřen na optimalizaci modelů signálů. Druhý a třetí typ algoritmu jsou si podobné, nicméně jejich implementace v testovací aplikaci vyžadovala odlišný způsob, proto bude každý popsán samostatně. Jedná se o algoritmy na vyhledávání specifických úseků a porovnávání shodných úseků.

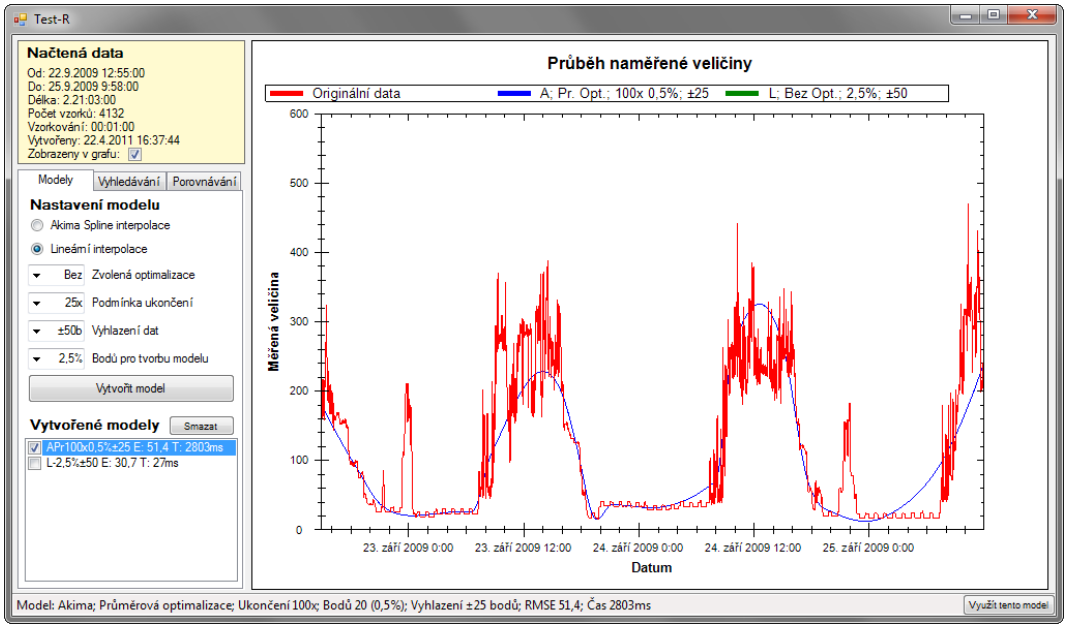
3.4.1 Všeobecný popis aplikace

Ačkoli se jedná pouze o testovací aplikaci, dbal jsem na určitou grafickou úpravu a také jsem se zaměřil na uživatelsky přívětivé chování programu. Snažil jsem se zamezit uživatelským chybám např. deaktivací tlačítek, kdy není možné danou funkci provést. Okno testovací aplikace je znázorněno na obrázku 3.7.

Pro celý program vystačí jedno okno, které je rozdělené do tří částí. V celé aplikaci je dominantní plocha grafu, kde je znázorněn průběh originálních dat, dále se zde zobrazují jednotlivé modely a znázorňují jednoltivé výsledky vyhledávacích algoritmů. V neposlední řadě slouží plocha grafu k označení části hledaného průběhu. Graf je vykreslován pomocí komponenty ZedGraph, se kterou jsem pracoval již během bakalářské práce. Z tohoto důvodu jsem ji použil i v testovací aplikaci.

Druhým významným prvkem je TabControl obsahující tři záložky. Každá z nich představuje jeden typ testovaného algoritmu a obsahuje patřičné nastavovací parametry a prostor pro zobrazení výsledků.

Zbytek okna testovacího programu je vyhrazen pro výpis informací o načtených originálních datech. Je zde informace o čase, ve kterém byla data měřena, celkový počet vzorků a doba vzorkování. Pod těmito infomacemi je zaškrtávací pole, pomocí kterého lze kdykoli zobrazit či schovat originální průběh v grafu.



Obr. 3.7: Okno testovací aplikace

3.4.2 Modelování signálu

Hlavním účelem testování modelovacích algoritmů je určit jejich rychlost a Přestože modelovací algoritmy dokáží pracovat se všemy typy interpolací, bylo zřejmé již v průběhu jejich vytváření, že mezi nejpoužívanější se zařadí akima spline interpolace a lineární interpolace. Z tohoto důvodu jsem ostatní typy do testovací aplikace nezahrnul.

Typy modelovacích algoritmů

Prvním modelovacím algoritmem, který jsem vytvořil, bylo modelování bez optimalizace, tedy s rovnoměrně rozdělenými body popisujícími výslednou křivku. Tento algoritmus jsem do testovací aplikace zahrnul hned z několika důvodů. Prvním z nich je jeho rychlost, která mnohonásobně převyšuje optimalizované modelovací algoritmy. Druhým důvodem jeho zařazení je skutečnost, že při stejných parametrech a datech dává pokaždé stejný výsledek. Je tedy možné tento model považovažovat za určitý základ, ke kterému je možné vztáhnout výsledky optimalizovaných modelů.

Optimalizované modely jsou založeny na genetickém algoritmu. Celkem jsem naprogramoval čtyři odlišné varianty a všechny jsem zahrnul do testovací aplikace. Cílem je zjistit, zda je mezi nimi určitý rozdíl v rychlosti, případně v kvalitě výsledných modelů.

Volitelné parametry

Vytvářený model lze nastavit pomocí pětice parametrů. Pvním z nich je typ interpolační křivky. Jak již bylo uvedeno výše, na výběr je lineárni či akima spline. Druhým parametrem je typ optimalizace. Zde je možné vybírat ze čtyř typů genetického algoritmu, případně zvolit model bez optimalizace. Následuje parametr podmínky ukončení optimalizace. Zde se vybírá hodnota, kolikrát genetický algoritmus provede optimalizaci bez zlepšení, než dojde k jeho zastavení. Větší hodnota prodlužuje dobu optimalizace na úkor možnosti dosažení lepších výsledků. Následujícím parametrem se volí vyhlazení vstupních dat před samotným vytvářením modelu. Nastavuje se velikost okolí, z kterého bude vypočten aritmetický průměr pro příslušný bod. Samozřejmostí je možnost volby bez vyhlazení. Posledním parametrem je celkový počet bodů, který bude tvořit výsledný model. Hodnota je udávána procentuálně a zvolený počet bodů se tak odvíjí od počtu vzorků v celém naměřeném signálu.

Práce s modely

Po nastavení jednotlivých parametrů je možné pomocí tlačítka vytvořit požadovaný model. Po jeho dokončení se průběh namodelovaných dat zobrazí v grafu a spolu s krátkým popiskem také v seznamu vytvořených modelů. Popisek obsahuje i zaškrtávací pole, pomocí kterého je možné průběh daného modelu v grafu zviditelnit či nikoli. Po výběru jednoho modelu v seznamu vypíše aplikace ve spodní části okna kompletní informace o zvoleném modelu. Jedná se o parametry, se kterými byl model vytvořen, o odchylku od skutečných dat a o čas vytváření modelu. Jelikož ostatní testované metody umožňují pracovat s namodelovanými daty, obsahuje informační lišta také tlačítko, které slouží k uložení vybraného modelu do paměti a je možné jej dále v programu využívat.

3.4.3 Vyhledávání specifických úseků

Algoritmus na vyhledávání specifických úseků má za úkol nalézt v celých datech všechny výskyty uživatelem definovaného specifického průběhu. Celkem jsem k tomuto účelu vyvinul dva podobné algoritmy, které již byly podrobně popsány dříve v textu. Hlavním účelem testovací aplikace je vyhodnotit kvalitu vyhledávání. Druhým sledovaným parametrem je čas vyhledávání.

Volitelné parametry

Průběh vyhledávání lze ovlivnit třemi parametry a pochopitelně výběrem hledaného specifického úseku. Prvním parametrem se volí, zda bude vyhledávání prováděno přímo v naměřených datech, nebo v modelovaném průběhu. Pomocí druhého parametru lze vybrat typ vyhledávacího algoritmu. Poslední možností, jak ovlivnit výsledky vyhledávání, je nastavení prahu. Výběr samotného hledaného úseku je řešen přímým označením části dat v grafu.

Práce s vyhledáváním

V prvé řadě je nutné určit, zda se má vyhledávání provádět přímo v naměřených datech, nebo v modelu. První volba je možná vždy, ale pro vyhledávání v modelu je nutné nejdříve tento model vytvořit. Popis uložení modelu do paměti programu bylo popsáno výše. Po aplikování tohoto postupu je tedy možné zvolit i vyhledávání v modelu. Aby byl uživatel informován, který model je aktuálně uložen v paměti, jsou jeho parametry vypsány v malém informačním panelu.

Výběr hledaného úseku se provádí myší přímo v grafu. Jelikož graf sám o sobě na stejný pohyb myší reaguje přiblížením zobrazovaných průběhů, je nutné před samotným vybíráním stisknout příslušné tlačítko. Tím se přibližování dočasně deaktivuje a uživateli je tak umožněno vybrat část dat. Vybraný úsek je vyznačen na pozadí za průběhy barevným obdélníkem. O vybraném úseku je uživatel informován stručným popiskem.

Po nastavení všech parametrů a výběru hledaného úseku je možné spustit vyhledávací algoritmus. Všechny nalezené shody jsou zobrazeny v seznamu, přičemž každá shoda je popsána datem začátku výskytu daného úseku a odchylkou od předlohy. Po výběru jedné shody v seznamu se v grafu zobrazí druhý obdélník, označující výskyt nalezeného úseku.

3.4.4 Porovnávání shodných úseků

Algoritmus na porovnávání shodných úseků má za cíl navzájem porovnat stejné časové úseky. Vstupem do tohoto algoritmu jsou data již rozdělená na jednotlivé části, které se navzájem porovnávají. Zvolil jsem toto řešení, protože předpokládám, že při případném nasazení algoritmu v praxi, by rozseparování signálu řešila externí funkce. V testovací aplikaci bylo právě rozseparování dat nejkomplikovanější.

Rozdělení dat

Z důvodu zjednodušení problematiky jsem se v testovací aplikaci zaměřil pouze na porovnávání jednotlivých hodin a dnů, protože během vývoje programu nebyla k dispozici tak rozsáhlá data. U porovnávání týdnů by bylo navíc obtížné hledat v datech pondělky jako začátky týdenních úseků a u porovnávání měsíců by byl problém s odlišným počtem dnů v každém měsíci.

Již při spouštění aplikace jsou do testovacího programu předána načtená data z databáze. Ihned poté se vyhodnotí některé parametry těchto dat a spolu s tím se určí celkový počet hodin a dnů k vzájemnému porovnání. Tyto hodnoty se nerovnají celkovému počtu hodin či dnů v průběhu, proto je obtížné je určit. Je to dáno skutečností, že porovnávací funkce má za úkol vzájemně porovnávat jednotlivé celé časové úseky. Hodinové úseky tedy musí začínat v každou celou hodinu a denní úseky o půlnoci. Ve většině případů tak dochází k situaci, že celkový počet možných porovnávaných úseků je menší, než celkový počet těchto úseků v datech.

Druhá část separace dat se provádí až před samotným spuštěním porovnávacího algoritmu, jenž vyžaduje jako vstupní parametr pole jednotlivých průběhů

porovnávaných úseků. Je tedy nutné z originálních dat toto pole vytvořit. Využívá se přitom hodnoty vypočtené po načtení dat. Ze známého vzorkování originálního signálu se určí počet vzorků v jednotlivých porovnávaných úsecích. Z těchto údajů se následně vytvoří požadovaná struktura dat.

Volitelné parametry

Porovnávací algoritmus je možné nastavit pomocí čtyř parametrů. Prvním parametrem se volí stejně jako u předešlé funkce, zda bude porovnávání prováděno v naměřených datech, nebo v modelovaném průběhu. Druhým parametrem je výběr porovnávaného úseku. Jak již bylo uvedeno, k dispozici jsou hodinové, či denní úseky. Aby byl uživatel informován o jejich celkovém počtu, je za každou volbou v závorce uvedena příslušná hodnota. Třetím parametrem je typ metody, kterou bude porovnávání prováděno. Na výběr jsou čtyři různé způsoby. Poslední parametr se týká volby prahu, do kterého se porovnávané úseky považují za shodné.

Práce s porovnáváním

Porovnávání je možné provádět na originálních či modelovaných datech stejně, jako je tomu u vyhledávacího algoritmu. Ovládací prvky i princip výběru je shodný s předešlou funkcí, proto není nutné ji zde znovu popisovat. Po následující volbě délky úseku, metody a prahu je možné příslušným tlačítkem spustit porovnávací algoritmus. Výsledky jsou spolu s informačním textem vypsány do seznamu. Informační text obsahuje časy začátků nalezených shodných úseků a hodnotu jejich vzájemné odchylky. V informační liště je spolu s tím vypsána informace o proběhlém algoritmu.

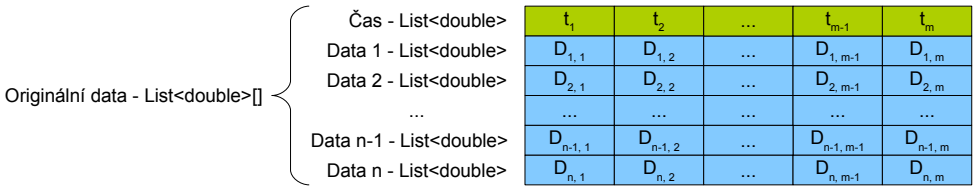
Po výběru jedné shody ve výsledném seznamu se podobně jako u předešlé metody objeví na pozadí grafu dva barevné obdélníky, které označují umístění vybraných shodných úseků v daném výsledku.

3.5 Struktury dat

Pojmen struktura dat není myšlena přímo struktura používaná v programovacím jazyce. Tímto pojmen označuji blok dat, se kterými se v aplikaci pracuje. Většina těchto struktur je typu pole listů. Listy v převážné většině obsahují neceločíselné hodnoty. Dále se v programu vyskytují data uložená v klasickém jednorozměrném poli. Popis jednotlivých struktur uvádím z důvodu nekorespondence indexů mezi jednotlivými datovými strukturami.

3.5.1 Originální data a průměrovaná data

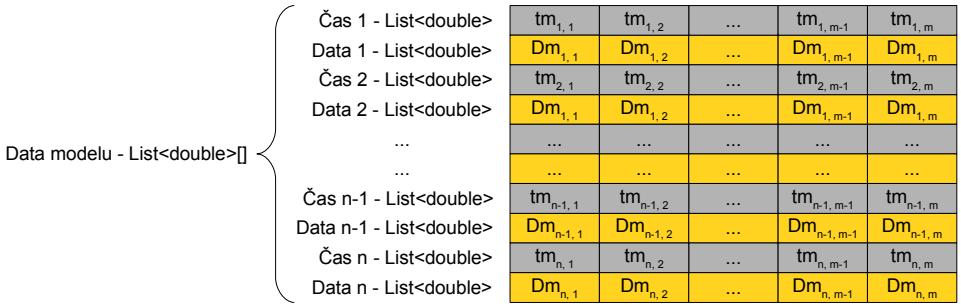
Struktura s originálními daty je základní skupina dat, která se v programu využívá. Ve většině případů vzniká načtením dat z databáze. Strukturu dat tvoří pole listů obsahující neceločíslené hodnoty. Tato skupina dat se vyznačuje tím, že obsahuje jeden list s hodnotami času a dále alespoň jeden list s daty. Listů s daty může být teoreticky neomezené množství. Vždy musí platit, že všechny listy v poli mají stejnou délku, to znamená, že obsahují stejný počet hodnot. Průměrovaná data mají naprosto shodné uspořádání dat, jako originální data, a vznikají jejich vyhlazením příslušnou funkcí.



Obr. 3.8: Struktura uložení originálních dat

3.5.2 Model

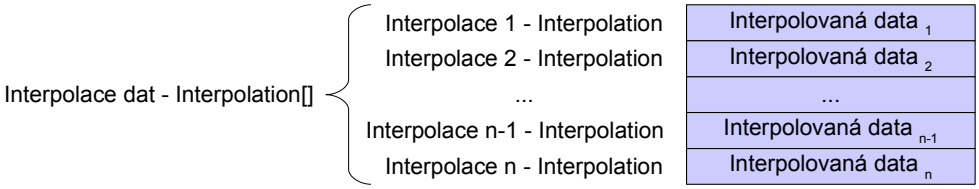
Model vzniká vždy z originálních či průměrovaných dat. Při jeho tvorbě je zapotřebí struktura dat uvedená výše. Struktura model je tvořena polem listů. Počet listů v poli se přímo odvíjí od počtu listů ve vstupních datech a rovná se dvojnásobku listů s daty. Je to dáno tím, že v modelu nestačí jeden společný list s hodnotami času pro všechny datové průběhy, ale každý list s daty potřebuje vlastní list s hodnotami času. Tato skutečnost vyplývá z vytváření optimálních modelů jednotlivých datových řad, kdy je nutné pro každou řadu vybrat jiné body. Platí však, že okrajové hodnoty času v modelu odpovídají vždy první a poslední hodnotě času z originálních dat. Je to dáno tím, že používané interpolační metody jsou schopny interpolovat jen hodnoty v zadaném rozsahu dat. Toto pravidlo musí být dodrženo, aby bylo možné data interpolovat v celém rozsahu.



Obr. 3.9: Struktura uložení dat modelu

3.5.3 Interpolace

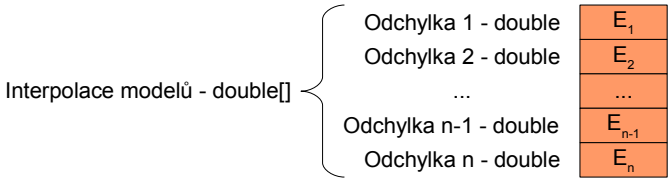
Po vytvoření dat modelu je nutné získat jednotlivé interpolované průběhy. K tomuto účelu slouží pole interpolací. Jeho velikost je rovna polovině listů ve struktuře modelu. Je to dané tím, že do každé interpolace vstupuje čas a data, a výstupem je jeden interpolovaný průběh. Zajímavé je, že pro interpolaci postačí pouze obyčejné pole, protože veškeré interpolované hodnoty obsahuje samotná interpolace. Vytvořených interpolací se následně stačí dotazovat na hodnoty v požadovaných časech.



Obr. 3.9: Struktura uložení interpolací

3.5.4 Kvadratická odchylka

Každé interpolaci odpovídá určitá kvadratická odchylka. Proto je uspořádání kvadratických odchylek podobné uspořádání interpolací. Jedná se opět o jednorozměrné pole neceločíselných hodnot. Kvadratická odchylka se počítá pomocí funkce, do které vstupuje interpolace a z originálního průběhu čas a příslušná data. Rozměr pole s kvadratickými odchylkami musí mít stejný rozměr jako pole interpolací.

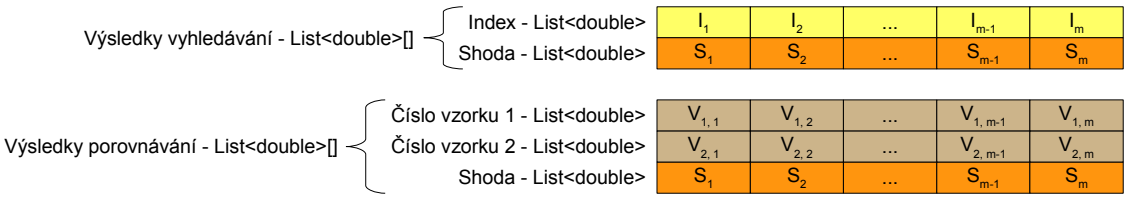


Obr. 3.10: Struktura uložení odchylek modelů

3.5.5 Výsledky vyhledávacích algoritmů

Vyhledávací algoritmus vrací strukturu dat tvořenou dvoupoložkovým polem listů. V prvním listu jsou uloženy hodnoty indexů ukazující na začátky nalezených shodných úseků. Indexy odpovídají pozicím v datech, ve kterých bylo vyhledávání prováděno. Druhý list obsahuje míry shod mezi vzorem a nalezenou částí signálu.

Druhý algoritmus na porovnávání shodných úseků dat vrací obdobné pole, které je však tvořeno třemi listy. První dva obsahují indexy ze vstupních porovnávaných dat. Ty označují nalezené podobné úseky. Třetí list obsahuje stejně jako u předešlé metody hodnotu udávající vzájemnou shodu obou průběhů.



Obr. 3.11: Struktura výsledků vyhledávacího a porovnávacího algoritmu

3.6 Pomocné funkce a algoritmy

Do této skupiny řadím veškeré funkce, které přímo nesouvisejí se zadanou problematikou, ale výrazně ulehčují vývoj hlavních algoritmů. Za nejvýznamnější funkci, spadající do této kategorie, bych zařadil generátor náhodného signálu. Jeho význam spočívá především v tom, že při vytváření programu není nutné neustále přistupovat do databáze a načítat z ní data.

Úkolem generátoru je vytvořit signál přibližně stejné podoby, jaký lze načíst z databáze, kde jsou uloženy naměřené hodnoty. Využití nejjednoduššího postupu, při kterém se každý vzorek signálu generuje zcela náhodně, tedy není možné. Je potřeba komplikovanějšího postupu. Nejdříve se vygenerují zcela náhodné hodnoty pro určitý počet bodů generovaného signálu. Tyto body jsou od sebe vzdáleny rovnoměrně. Následně se tyto body proloží spojitou křivkou a pomocí interpolace se určí zbývající body průběhu. Takto hladký signál však stále neodpovídá reálnému průběhu. Z tohoto důvodu se v posledním kroku každý bod posune o náhodnou vzdálenost směrem nahoru či dolů. Ačkoli je tato vzdálenost náhodná, je omezena její maximální hodnota, aby body zůstaly stále poblíž interpolované křivky.

Následující důležitou funkcí je algoritmus na nalezení nejideálnějšího modelu. Z důvodu velké výpočetní náročnosti ji však nelze aplikovat na reálná data. Algoritmus totiž postupně zkouší všechny možné kombinace výběru bodů pro model. Každý model se vytvoří a spočítá se jeho odchylka od skutečných dat. Výsledkem je kombinace nejlepších bodů pro definování modelu. Řešení všech kombinací je klíčovou slabinou celého algoritmu, protože i s malým nárůstem délky modelovaného signálu nebo počtem bodů definujících model, výrazně roste počet všech kombinací. Jako příklad lze uvést, že již pro interpolaci signálu o délce 150 vzorků s definováním modelu pomocí pouhých 10 bodů se jedná přibližně o 10^{15} kombinací. Přínosem této funkce je však možnost porovnávání výsledků optimalizačních algoritmů již během jejich vývoje. K tomuto účelu jsem používal signál s 26 vzorky a model tvořilo 6 bodů.

4 Dosažené výsledky

4.1 Dokončené funkce, algoritmy a programy

Mezi první velikou skupinu vytvářených algoritmů spadají veškeré funkce na vytváření modelů. Tuto skupinu lze rozdělit na optimalizované a neoptimalizované algoritmy. Druhá jmenovaná skupina vznikla hned v úvodu diplomové práce a po celou dobu vývoje optimalizovaných funkcí sloužila jako reference, zda je vývoj optimalizačních metod v pořádku a skutečně se jedná o optimalizaci. Právě vzájemná porovnávání obou metod na tvorbu modelů ukázala, že první dvě experimentálně vyvinuté optimalizace nedosahují požadovaných výsledků. Jedná se o algoritmy využívající umístění bodů modelu do lokálních maxim a minim a optimalizace založená na postupném posunu jednotlivých bodů. Z tohoto důvodu vznikla čtveřice metod založených na genetické optimalizaci. Jejich výsledky se zdály být během vývoje uspokojivé, a proto jsem je zařadil k otestování v závěrečném testovacím programu. Dále jsem se zabýval modelem horní a dolní obálky signálu. Výsledné algoritmy fungují, nicméně výsledný průběh není příliš hladký, a proto se zde nabízí určitá možnost dalšího vylepšování.

Druhou významnou skupinou algoritmů jsou funkce na vyhledávání, případně určování podobnosti signálů. Celkem jsem vytvořil šest funkcí, z nichž dvě jsou určené pro vyhledávání zadaných signálů. Zbývající čtyři funkce slouží k vzájemnému porovnávání shodných úseků. Části algoritmů jsou si navzájem podobné, v některých částech i shodné. Navzájem se od sebe odlišují použitým parametrem pro rozhodnutí o shodě. Všechny vyhledávací metody jsou plně funkční a jejich výsledná kvalita byla otestována v testovací aplikaci.

Poslední významnou částí diplomové práce je samotný testovací program. Jako jediný má grafické rozhraní a slouží k jedinému účelu, k otestování vytvořených algoritmů. Na základě zjištěných výsledků je možné vyhodnotit kvalitu jednotlivých funkcí a rozhodnout, zda jsou některé postupy vhodné pro další použití či nikoli. I přes svoji pouze testovací funkci jsem výsledný program vytvářel tak, jako by měl být standardně používán. To znamená, aby bylo ovládání jednoduché, uspořádání jednotlivých prvků upravené a zobrazení výsledků přehledné.

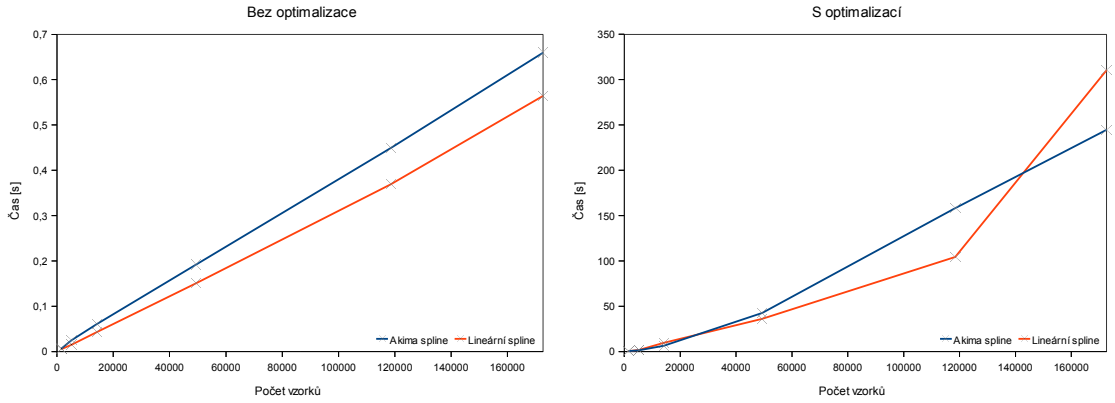
4.2 Testy algoritmů na vytváření modelů

4.2.1 Rychlost vytváření modelů

Testování rychlosti vytvář0ení modelů jsem prováděl na vlastním PC. Žádné speciální podmínky, jako například čistě nainstalovaný operační systém, jsem nevytvářel. Hlavním cílem testů není určení absolutních hodnot rychlostí, ale spíše vzájemné porovnání na jednotlivých parametrech. Všechna měření jsem zopakoval pětkrát a do grafů vynášel výsledný průměr z těchto hodnot.

Rychlost vytváření modelu v závislosti na počtu bodů originálních dat

Prvním zkušebním testem, který jsem provedl, byla závislost rychlosti vytváření modelů na počtu bodů originálních dat, ze kterých je model získán. Celkem jsem provedl čtyři druhy měření. První dvě měření byla zaměřená na neoptimalizované metody, tedy na lineární a akima spline interpolaci. Druhé dvě porovnávají rychlost optimalizovaných metod, konkrétně pomocí dvoubodové genetické optimalizace. Všechny ostatní parametry během testů byly shodné pro všechna měření. Model byl tvořen 2,5% bodů, neprovádělo se žádné vyhlazení dat a u optimalizovaných metod byla podmínka ukončení nastavena na hodnotu 10.

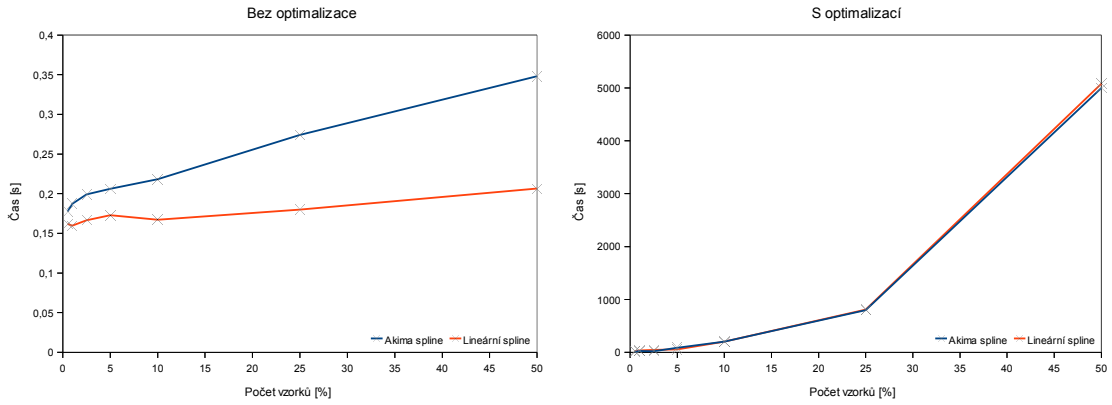


Graf. 4.1: Rychlost vytváření modelu v závislosti na počtu bodů originálních dat

Výsledky nejsou příliš překvapivé. U neoptimalizovaných metod vytváření modelů roste časová náročnost tvorby modelu lineárně. Jako rychlejší se jeví z těchto dvou metod lineární interpolace. U optimalizovaných metod není měřená závislost lineární, ale spolu se zvyšujícím se počtem bodů v originálních datech se také zvyšuje časová náročnost tvorby modelu. Výsledky měření jsou zobrazeny v grafu 4.1, který je sestrojen z hodnot v tabulce A.1 v příloze.

Rychlost vytváření modelu v závislosti na počtu bodů tvořících model

V následujícím testu jsem se zaměřil na rychlost vytváření modelu v závislosti na počtu bodů tvořících model. Provedl jsem celkem čtyři měření, z nichž dvě byla pro otestování neoptimalizovaných modelů a dvě pro optimalizované metody. Pro měření jsem vybral data s 49 299 vzorky. Tato hodnota však mohla být menší, protože vytváření optimalizovaných modelů s 50% bodů z originálních dat trvalo přibližně 80 minut. Ostatní parametry během testu byly nastaveny podobným způsobem jako u předešlé zkoušky. Neprovádělo se žádné vyhlazování dat, prováděla se dvoubodová optimalizace a ukončení bylo nastaveno na hodnotu 10.

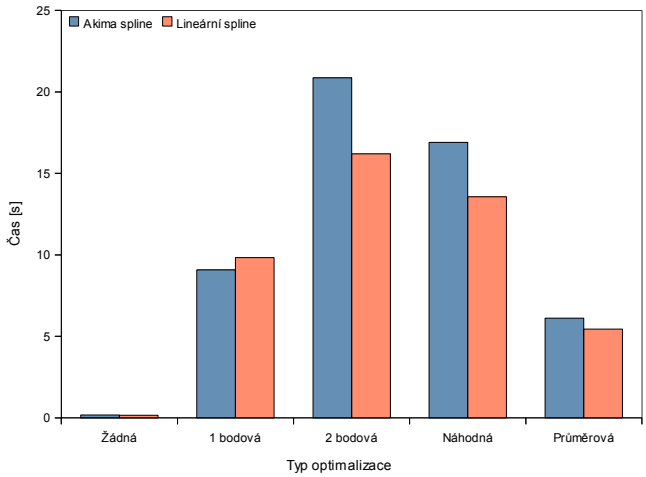


Graf. 4.2: Rychlost vytváření modelu v závislosti na počtu bodů tvořících model

Měřená závislost u neoptimalizovaných modelů vykazuje lineární závislost. Stejně jako v předchozím testu, i zde je lineární interpolace rychlejší. Oproti tomu optimalizované metody s přibývajícím počtem bodů tvořících model vyžadují mnohem více času na výpočet. U obou typů interpolace vychází přibližně shodný průběh. Výsledek je v celku pochopitelný, protože s přibývajícím počtem bodů výrazně stoupá výpočetní náročnost optimalizačního algoritmu. Výsledky jsou k vidění v přiloženém grafu 4.2. Tabulka A.2 je umístěna v příloze.

Rychlost vytváření modelu v závislosti na typu optimalizace

Cílem testování závislosti rychlosti na typu interpolace je snaha určit, zda jsou některé optimalizační algoritmy výrazněji rychlejší. Pro porovnání jsem zařadil i rychlost vytváření modelů bez optimalizace. Testování probíhalo, jak na modelech tvořených lineární, tak akima spline. Jako zkušební data jsem vybral signál o 49 299 vzorcích. Modely byly vytvářeny z 0,5% bodů v originálních datech. Neprovádělo se žádné vyhlazení a hodnota ukončení byla nastavena na 10.

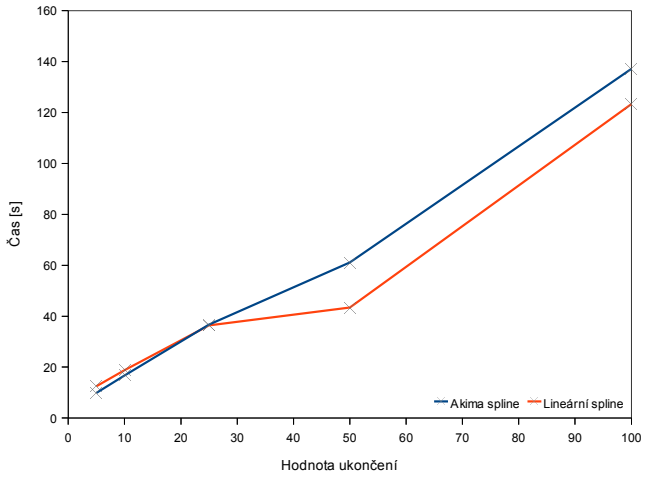


Graf. 4.3: Rychlost vytváření modelu v závislosti na typu optimalizace

Výsledky ukazují, že nejrychlejším optimalizačním algoritmem je průměrovací algoritmus. Následuje ho jednobodová optimalizace. Nejpomalejší je dvoubodová optimalizace. Pořadí rychlostí optimalizačních metod vychází pro oba typy modelů stejné, celkově je však lineární interpolace celkově rychlejší. Výsledky jsou názorně zobrazeny v grafu 4.3 , případně v tabulce A.3 v příloze.

Rychlost vytváření modelu v závislosti na ukončení

Na rychlost vytváření modelu má pochopitelně vliv také nastavená hodnota ukončení. Tento test má za úkol tuto závislost určit. Pro zkoušku byly použity data o 49 299 vzorcích, model tvořilo 0,5% bodů. Byla použita dvoubodová optimalizace bez předchozího vyhlazení dat.



Graf. 4.4: Rychlost vytváření modelu v závislosti na ukončení

Délka vytváření modelu je podle naměřených výsledků ve své podstatě lineární závislostí. Odchylky jsou podle mého názoru způsobeny pouze pětinásobným opakováním měření. Domnívám se, že při větším počtu opakování by odchylky od linearity nebyly tolik patrné. Graficky jsou výsledky znázorněny v grafu 4.4 a naměřené hodnoty jsou zaneseny v tabulce A.4 v příloze.

Rychlost vytváření modelu v závislosti na typu interpolační křivky

Z předešlých testů je patrné, že modely tvořené lineární interpolací jsou o něco rychlejší. Z důvody shodného použití obou typů interpolace, jak již v optimalizovaných či neoptimalizovaných metodách, je patrné, že tento fakt bude způsoben samotným algoritmem na vytváření modelů v knihovně Math.NET.

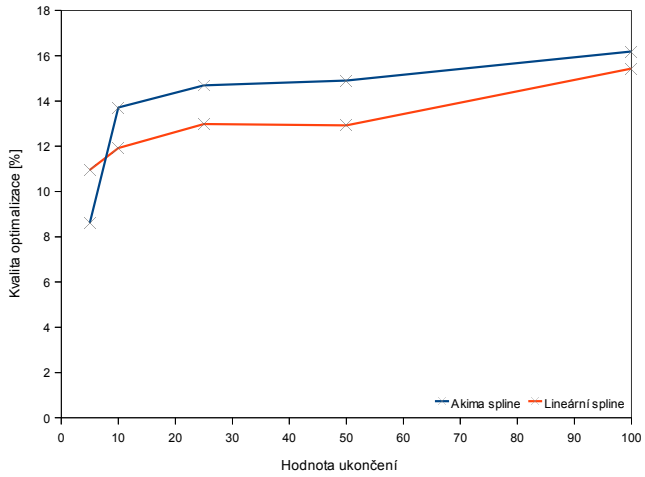
4.2.2 Porovnání kvality výsledných modelů

V případě, kdy je na vytvoření modelu k dispozici dostatek času, je hlavním parametrem jeho kvalita. Kvalitu modelu lze posuzovat podle průměrné kvadratické odchylky. Hlavním cílem optimalizačních metod je zlepšení kvality modelu. Hodnota průměrné kvadratické odchylky neoptimalizovaných modelů je tedy v následujících testech považována za základ, od kterého se určuje kvalita optimalizace. Každé měření je provedeno pětkrát z důvodu lepší věrohodnosti výsledků.

Kvalita modelu v závislosti na ukončení optimalizace

Prvním testem zaměřeným na kvalitu výsledných optimalizovaných modelů je určení závislosti kvality modelu na délce ukončení. Hodnota ukončení udává, kolikrát se bezúspěšně opakuje jeden krok genetického algoritmu, než dojde k jeho celkovému ukončení. Otestoval jsem oba typy modelů. Pro lepší názornost výsledků jsem nejdříve vytvořil neoptimalizovaný model, jeho chybu vzal jako referenční hodnotu a vůči ní následně vyjadřoval procentuální zlepšení optimalizovaných modelů. Pro tento test jsem zvolil dvoubodovou optimalizaci, nepoužil vyhlazení dat a počet bodů modelu jsem nastavil na 0,5% z celkového počtu vzorků v originálních datech.

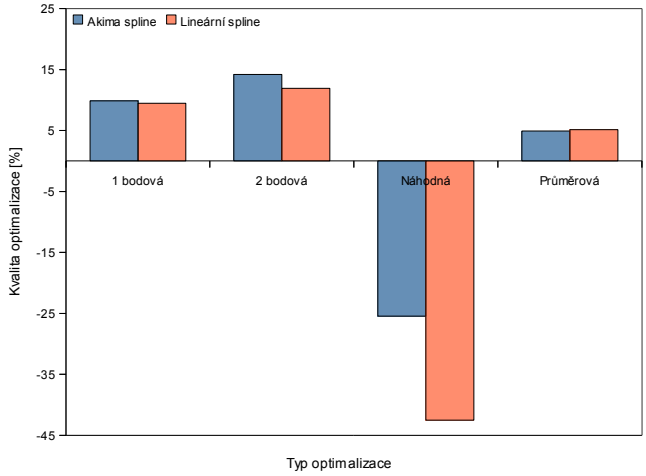
Výsledky testu ukazují, že nejhorších výsledků je dosaženo pro malé hodnoty ukončení, protože optimalizační proces má k dispozici malý počet iteračních kroků pro možné zlepšení. Od hodnoty ukončení přibližně 25 se při zvětšování tohoto parametru kvalita modelu zlepšuje jen mírně. To je možné vysvětlit přiblížením se k maximálnímu možnému nejlepšímu modelu, kde není mnoho prostoru pro výraznější zlepšování. Výsledky jsou znázorněny v grafu 4.5, případně v tabulce B.1 v příloze.



Graf. 4.5: Kvalita modelu v závislosti na ukončení optimalizace

Kvalita modelu v závislosti na typu optimalizace

Následujícím testem na zhodnocení kvality je vzájemné porovnání jednotlivých optimalizačních metod. Stejně jako v minulém případě, i v tomto testu jsem jako referenci použil hodnotu neoptimalizovaného modelu a výsledky vyjádřil procentuálně. Měření probíhalo na datech o 49 299 vzorcích. Model byl tvořen 0,5% bodů z této hodnoty. Nebylo provedeno žádné vyhlazení a ukončení bylo nastaveno na hodnotu 50.



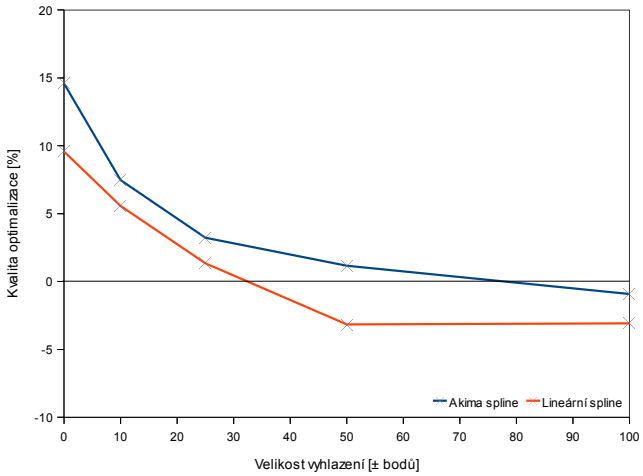
Graf. 4.6: Kvalita modelu v závislosti na typu optimalizace

Zajímavým výsledkem tohoto testu je zajisté náhodná optimalizace, která v porovnání s neoptimalizovaným modelem dosáhla dokonce horších výsledků. Je tedy patrné, že pro praktické použití je tato metoda nepoužitelná. Zbylé tři metody dosáhly vůči neoptimalizovanému modelu určitého zlepšení. Nehorší z této trojice vyšel

průměrovací optimalizační algoritmus, naopak nejlepších výsledků dosáhl algoritmus dvoubodový. Výsledky jsou znázorněny v grafu 4.6 a v příloze v tabulce B.2.

Kvalita modelu v závislosti na velikosti vyhlazení dat

Dále jsem se zaměřil na vyhodnocení kvality vytvářených modelů v závislosti na vyhlazení originálních dat. Model se tedy vytváří z takto předzpracovaných hodnot, nicméně výsledná chyba modelu se stále určuje vůči originálnímu nevyhlazenému průběhu. Z důvodu, že vyhlazení dat ovlivní i neoptimalizovaný model, jsem využil během tohoto testu pro každé vyhlazení odlišnou referenční hodnotu. Ta je dána chybou neoptimalizovaného modelu s patřičným vyhlazením. Během testu byla použita dvoubodová optimalizace s ukončením nastaveným na hodnotu 50. Model tvořilo 0,5% bodů.



Graf. 4.7: Kvalita modelu v závislosti na velikosti vyhlazení dat

Výsledky jsou vcelku zajímavé, protože spolu se zvětšujícím se vyhlazením dat klesá chyba neoptimalizovaného modelu. Při větších hodnotách vyhlazení nemůže optimalizace dosáhnout takového zlepšení kvality modelu jako při modelování nevyhlazených dat. Při velkých hodnotách vyhlazení optimalizované modely dávají dokonce horší výsledky než modely neoptimalizované. Domnívám se, že je to způsobeno značným vyhlazením, které neumožňuje tvorbu modelu z vhodných bodů. Výsledky se shodují pro obě interpolační metody a jejich průběhy jsou znázorněny v grafu 4.7 a v tabulce B.3 v příloze.

Kvalita modelu v závislosti na typu interpolační křivky

Dle předchozích testů lze usuzovat na mírně lepší výsledky u modelů tvořených akima spline interpolací. Nicméně všechny testy probíhaly na stejných datech a je tedy možné, že pro jiná data by vyšla kvalita jednotlivých interpolací odlišně. Rozdíly jsou navíc velmi malé, proto usuzuji, že oba typy modelů jsou rovnocenné. Při používání modelů v praxi by tak byl rozhodujícím faktorem účel následného použití modelu.

4.3 Testy algoritmů na vyhledávání a porovnávání úseků dat

4.3.1 Rychlost vyhledávacích a porovnávacích algoritmů

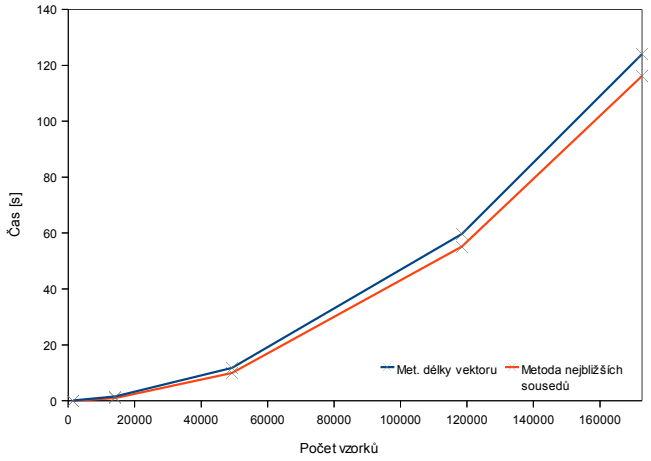
Testování rychlosti vyhledávacích algoritmů jsem prováděl obdobně jako měření rychlosti vytváření modelů. Z důvodu, že vyhledávací a porovnávací algoritmy jsou vytvořené na porovnávání stejně dlouhých úseků, kde se předpokládá rovnoměrné rozložení vzorků, nerealizoval jsem rozdílné vyhledávání v modelu a v originálních datech. Model je popsán určitým počtem bodů z originálních dat a typem křivky, nicméně v programu se interpolují všechny body na pozicích originálních dat. Proto jsem při testech rychlosti vyhledávání využíval pouze originálních dat.

U měření rychlosti porovnávacích algoritmů jsem byl omezen daty, na kterých jsem testy prováděl. Jelikož se vzájemně porovnávají pouze celé hodiny a dny v načtených datech a navíc vzorkování každých dat je odlišné, není možné tyto parametry volitelně měnit. Je tak nemožné získat požadované množství relevantních měření k tomu, aby vznikla kvalitní charakteristika. Z tohoto důvodu jsem se u porovnávacího algoritmu zaměřil pouze na otestování rychlosti jednotlivých porovnávacích metod.

Rychlost vyhledávání v závislosti na počtu bodů originálních dat

Doba vyhledávání pochopitelně závisí na délce vstupních dat a na délce hledaného intervalu. V tomto testu jsem zvolil délku vyhledávaného intervalu pevně na 100 bodů a zaměřil se pouze na závislost výpočetní náročnosti vyhledávacího algoritmu na počtu vzorků v datech. Na hodnotě prahu by neměla být doba vyhledávání závislá, nicméně v každém testu jsem nastavil hodnotu 0,3.

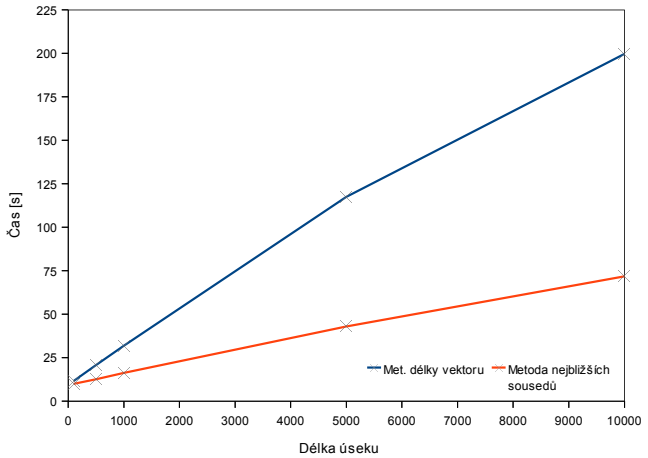
Naměřená závislost vychází pro oba vyhledávací algoritmy obdobně. Výpočetní doba však nenarůstá s počtem vzorků v originálním průběhu lineárně, nýbrž pro vyšší hodnoty se délka výpočtu více prodlužuje. Naměřený průběh je znázorněn v grafu 4.8 a hodnoty zapsány v příloze v tabulce C.1.



Graf. 4.8: Rychlost vyhledávání v závislosti na délce vstupních dat

Rychlost vyhledávání v závislosti na délce hledaného úseku

Tento test slouží k určení závislosti délky vyhledávání na délce hledaného úseku. Délkou hledaného úseku je myšlen počet vzorků, kterými je daný úsek definován. Se zvětšující se počtem vzorků v hledaném úseku se zmenšuje celkový počet porovnávání, který musí algoritmus provést. Naopak porovnávání dvojic je časově náročnější. Z tohoto hlediska je tento test zajímavý. Zkoušku jsem provedl na datech o 49 299 vzorcích.



Graf. 4.9: Rychlost vyhledávání v závislosti na délce hledaného úseku

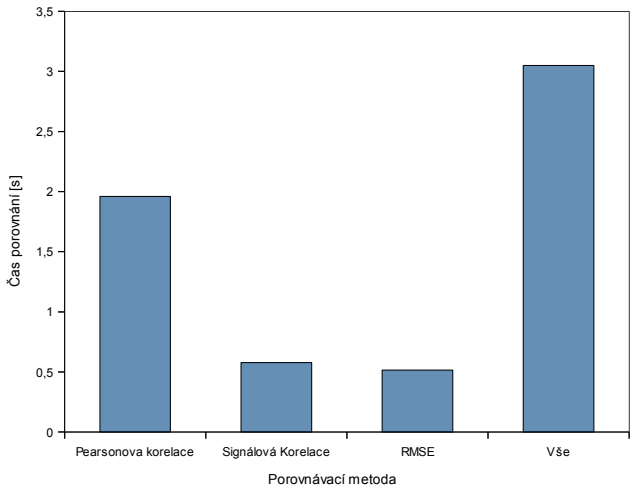
Výsledky jsou oproti výše uvedené skutečnosti velice zajímavé. Čas potřebný k dokončení vyhledávání je téměř lineárně závislý na délce hledaného úseku. Toto platí pro oba typy vyhledávacích algoritmů. Naměřené průběhy jsou zobrazeny v grafu 4.9, naměřené hodnoty v tabulce C.2 umístěné v příloze.

Rychlost vyhledávání v závislosti na zvolené metodě porovnávání

Z předešlých měření je patrné, že rozdíl v rychlosti jednotlivých algoritmů je tím větší, čím je delší hledaný úsek. Rychlejším vyhledávacím algoritmem je metoda nejbližších sousedů. Tato skutečnost je dobře vysvětlitelná, protože v této metodě není zahrnuto tolik výpočtů potřebných k vzájemnému porovnání signálů, jako u metody pracující s délkou vektoru.

Rychlost porovnávání v závislosti na zvolené metodě porovnávání

Některé problémy testování porovnávací metody byly již nastíněny výše, proto jsem provedl pouze jeden test. Jeho úkolem je navzájem porovnat rychlost jednotlivých porovnávacích metod. Celkově jsou tyto metody čtyři, nicméně poslední z nich je kombinací tří předešlých. Respektive využívá všechny předchozí metody. Z toho se dá již předem usuzovat, že právě tato poslední metoda bude trvat nejdéle. Samotný test jsem provedl na datech o 172 645 vzorcích. Porovnával jsem vzájemně jednotlivé dny, kterých bylo v takto rozsáhlých datech 119. Jeden den přitom obsahoval 1 440 vzorků.

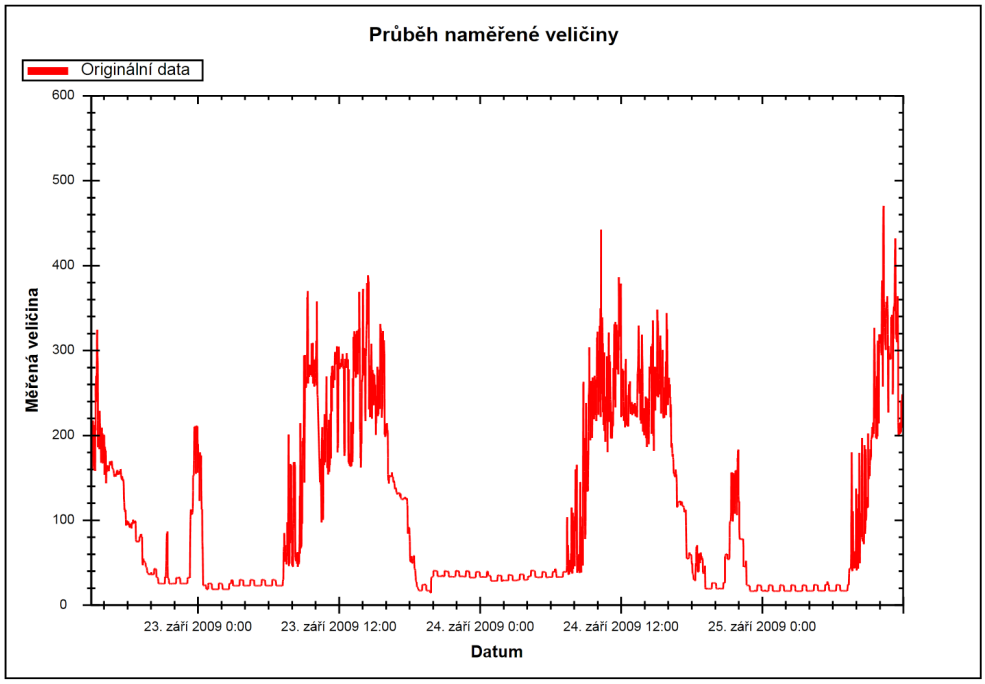


Graf. 4.10: Rychlost porovnávání v závislosti na zvolené metodě

Zajímavým výsledkem je metoda využívající k porovnání dvou signálů Pearsonova korelačního koeficientu. Na výpočet této hodnoty jsem použil funkci z knihovny Math.NET, zatímco výpočet signálové korelace, respektive průměrné kvadratické odchylky, jsem programoval sám. Přesto výpočet Pearsonova korelačního koeficientu trvá déle než mnou programované metody. Dalším zajímavým poznatkem je skutečnost, že algoritmus kombinující ostatní metody je časově přibližně stejně náročný, jako součet časových náročností jednotlivých algoritmů. Výsledky jsou zobrazeny v grafu 4.10 a v tabulce C.3 v příloze.

4.3.2 Kvalita vyhledávacích a porovnávacích algoritmů

Vyhodnocení kvality vyhledávacích a porovnávacích algoritmů je obtížněji realizovatelné, než vyhodnocování kvality modelů. Určení, zda je nalezená shoda určená správně či nikoli, nelze v testovacím programu měřit. Proto jsem provedl několik testů, pomocí kterých jsem, na základě svého úsudku, rozhodl o kvalitě daného algoritmu. Výslednou kvalitu jsem posuzoval na základě počtu správně a špatně porovnaných úseků.



Obr. 4.1: Průběhu použitý pro testování vyhledávacího a porovnávacího algoritmu

Kvalita vyhledávání v závislosti na zvolené metodě

Testování probíhalo na datech s průběhem, který je zobrazený na obrázku 4.1. Vyhledávání jsem začal testovat s krátkými úseky. Daných úseků bylo v originálním průběhu několik, což se pro danou zkoušku vyhovuje. Metoda založená na principu délky vektoru s prahem nastaveným na základní hodnotu 0,2 bezchybně označila všechna podobná místa. Oproti tomu metoda založená na principu nejbližších sousedů označila jen dva správné výsledky a jeden dokonce špatný.

Pro druhý test jsem vybral dlouhou výraznou část signálu, která se v originálních datech ještě jednou opakovala. Vybraný úsek byl však značně zašuměný oproti minulé zkoušce. V tomto případě metoda pracující s délkou vektoru nenašla žádnou shodu, zatímco metoda nejbližších sousedů správně označila podobný úsek. Avšak ani v tomto případě neproběhlo označení bez chyby a metoda označila opět jeden úsek navíc.

Z výsledků usuzuji, že metoda založená na délce vektoru je vhodná pro nezašuměný signál a pro kratší hledané úseky, jako jsou například krátkodobé výpadky, špičky a podobně. Naopak metoda nejbližších sousedů se jeví na vyhledávání tohoto druhu signálů naprosto nevhodná. Její použití se nabízí u delších časových úseků, které mohou být i mírně zašuměné.

Kvalita vyhledávání v závislosti na použitém modelu

Stejné testy jsem opakoval i v namodelovaném průběhu. Jednalo se o akima spline model, který byl vytvořen dvoubodovou optimalizací s ukončením 100 a vyhlazení dat mělo hodnotu 10. Jelikož model úplně vyhladil prvně vyhledávaný úsek, nebylo ani možné vyhledávací metody s touto krátkou částí dat testovat.

U druhého testu se prokázalo, že s využitím modelu, který ze signálu odstranil šum, je použitelná i metoda založená na délce vektoru, která označila správně pouze jeden výsledek. I druhá vyhledávací metoda dosáhla stejných bezchybných výsledků. U třetího, tedy kombinovaného testu dosáhly obě metody bezchybných výsledků, nicméně metoda pracující s délkou vektoru musela mít nastavený větší práh.

Na základě měření lze konstatovat, že vyhledávání v modelu namísto v originálních datech je vhodné pro velké úseky dat, které mohou být i zašuměné. Naopak není možné použít vyhledávání v modelu pro detekci krátkých, i když výrazných jevů, protože modelováním se tyto části znehodnotí.

Zhodnocení kvality porovnávání

Porovnávací algoritmus využívající všech parametrů pracuje na podobném principu jako vyhledávací metoda fungující na principu délky vektoru. Platí pro něj tedy podobná pravidla použití jako u výše testovaného vyhledávacího algoritmu.

Metoda porovnávání využívající Pearsonova korelačního koeficientu, nebo hodnoty signálové korelace, dosahuje kvalitních výsledků i v případě mírně zašuměných signálů. Metoda pracující s průměrnou kvadratickou odchylkou pro správné fungování naopak vyžaduje vyhlazený signál.

5 Závěr

V diplomové práci jsem se zabýval třemi odlišnými, přesto vzájemně se prolínajícími problematikami. Jednalo se o modelování signálu, které slouží jako předzpracování dat před aplikací dalších algoritmů. Mezi ně je možné zařadit detekci specifických opakujících se jevů. Výslednou částí je nasazení vytvořených postupů do klientské zkušební aplikace s následným využitím vytvořeného programu pro zhodnocení výsledků. Všechny cíle byly v průběhu diplomové práce úspěšně splněny.

Prvně řešenou problematikou byla otázka modelování signálů. V úvodu bylo vysvětleno, že se jedná o výběr optimálních bodů, které tvoří vstupní parametr funkce z využívané matematické knihovny. Hlavní otázkou se tak stala optimalizační úloha, která minimalizuje chybu modelu v závislosti na vybraných bodech tvořících model. Začal jsem se tedy zabývat genetickým algoritmem, který by umožnil zadaný problém řešit. Protože smyslem mé diplomové práce je vytvářet nové algoritmy, nesnažil jsem se implementovat již hotové postupy. Po nastudování základních principů, jak genetický algoritmus pracuje, jsem začal tvořit vlastní algoritmus zaměřený přesně na danou konkrétní problematiku. Po prvních testech, které prokázaly, že vytvořený optimalizační postup funguje, vznikly další tři varianty algoritmu. Při závěrečném testování se prokázalo, že všechny algoritmy úspěšně fungují, ale jeden z nich však nedosahuje požadovaných kvalit. Tento nedostatek byl objeven až v závěru práce. Nebylo tedy možné z časových důvodů daný optimalizační postup upravit. U funkčních algoritmů se nabízí možnost jejich dalšího vývoje. Vylepšení se nabízí především v oblasti rychlosti algoritmů, která v diplomové práci není nikterak optimalizována např. pro běh na více jádrových procesorech.

Ve druhé části diplomové práce jsem se zabýval problematikou detekcí specifických opakujících se jevů v signálu. Nalezení těchto úseků je nezbytnou podmínkou k tomu, aby bylo možné daný průběh určitým způsobem klasifikovat, např. z hlediska počtu výskytů daných jevů. Hlavním problémem tedy bylo rozpoznání vzájemně shodných či podobných signálů. K tomuto účelu jsem nastudoval určité postupy, které se v praxi běžně používají. Jejich základní principy jsem využil ve svých algoritmech, kterých jsem vytvořil celkem šest, z nichž některé jsou si podobné. Pro vyhodnocení shody mezi signály však využívají odlišných parametrů. Během závěrečného testování se ukázalo, že každý typ algoritmu je vhodný pro vyhledávání určitého typů úseků. Dále se prokázalo, že předchozí namodelování průběhu může

v některých případech velmi pozitivně ovlivnit výsledky vyhledávání. Stejně jako u optimalizačních metod, také u vyhledávacích algoritmů se nabízí možnost urychlení tohoto procesu. Další oblastí, kde vidím možnost určitého zlepšení, je přidání dalších parametrů, na jejichž základě se shoda signálů vyhodnocuje.

Poslední část diplomové práce byla více praktická, ve které jsem dříve vytvořené algoritmy implementoval do klientské aplikace. Pro realizaci tohoto úkolu bylo nutné navrhnout především uživatelské rozhraní, které by umožnilo nastavování a spouštění jednotlivých operací. Nezbytnou součást grafického rozhraní tvořilo také přehledné zobrazení výsledků a výpis důležitých informací. Poslední část diplomové práce ověřila, že vytvořené algoritmy mohou pracovat v reálné aplikaci. Jelikož se jedná o uživatelskou aplikaci, největší možnosti vylepšení se nabízejí právě v oblasti uživatelského rozhraní. V případě využití dané aplikace v praxi by bylo možné doplnit další volitelné parametry jednotlivých algoritmů, vypisovat podrobnější informace o získaných výsledcích nebo vytvořit vícevláknovou aplikaci, kde by bylo možné provádět více druhů výpočtů najednou.

K celkovému vyhodnocení výsledků jsem využil vlastní aplikaci a otestoval jednotlivé algoritmy. Provedl jsem řadu testů s následným posouzením kvalit a nedostatků jednotlivých testovaných metod. Na základě získaných informací mohu uvést, že všechny části diplomové práce byly splněny.

Z mého pohledu byla zajímavá především první část diplomové práce, kde se prolínala teoretická část s praktickou. Jednalo se o převod optimalizačních postupů do zdrojového kódu. Při tvorbě uživatelského programu bylo příjemné sledovat, že dříve vytvořené algoritmy začínají úspěšně pracovat také se skutečnými daty.

Seznam použité literatury

[1] *.NET Framework Developer's Guide*, [online],
URL: ⟨http://msdn.microsoft.com/⟩

[2] *ALGLIB User Guide – Interpolation and fitting*, [online],
URL: ⟨http://www.alglib.net/interpolation/⟩

[3] Christian Nagel: *C# 2008 Programujeme profesionálně*, Computer Press, Brno, 2009, ISBN: 978-80-251-2401-7

[4] *Dokumentace ke knihovnám Dxperience*, [online],
URL: ⟨http://www.devexpress.com/⟩

[5] *Dokumentace knihovny Math.NET*, [online],
URL: ⟨http://www.mathdotnet.com/doc/⟩

[6] Jaromír Špico: *Data mining v energetickém průmyslu*, Univerzita Tomáše Bati ve Zlíně, 2009, [online],
URL: ⟨http://dspace.knihovna.utb.cz/bitstream/handle/10563/8425/%C5%A1pico_2009_bp.pdf?sequence=1⟩

[7] KMB systems: *Manual SMV, SMP, SMPQ*, 2009, Liberec, [online],
URL: ⟨http://www.kmb.cz/07/content/view/51/36/⟩

[8] Lukáš Bajer. *Urychlení evolučních algoritmů pomocí směsí rozdělení pravděpodobnosti*, Univerzita Karlova v Praze, 2009, [online],
URL: ⟨http://artax.karlin.mff.cuni.cz/~bajel3am/download/dp.pdf/⟩

[9] Miroslav Virius: *C# – Hotová řešení*, Computer Press, Brno, 2006
ISBN: 80-251-1084-2

[10] Tomáš Svoboda, Michal Houdek, Tomáš Procházka: *Klasifikace podle nejbližších sousedů*, [online],
URL: ⟨http://cmp.felk.cvut.cz/cmp/courses/recognition/zapis_prednasky/zapis_01/4/rpz4.pdf⟩

Příloha A – Tabulky s výsledky testování rychlosti algoritmů na modelování dat

Akima spline - bez optimalizace

Počet bodů	1498	5163	14130	49299	118493	172645
t_1 [ms]	7	25	64	193	448	665
t_2 [ms]	7	25	59	213	449	659
t_3 [ms]	5	25	63	184	449	657
t_4 [ms]	7	25	64	184	447	658
t_5 [ms]	5	25	52	184	451	659
$t_ø$ [ms]	**6**	**25**	**60**	**192**	**449**	**660**

Lineární spline - bez optimalizace

Počet bodů	1498	5163	14130	49299	118493	172645
t_1 [ms]	6	15	43	151	367	543
t_2 [ms]	6	15	43	150	366	544
t_3 [ms]	6	15	42	150	367	645
t_4 [ms]	4	15	43	150	374	538
t_5 [ms]	4	15	42	151	371	551
$t_ø$ [ms]	**5**	**15**	**43**	**150**	**369**	**564**

Akima spline - s optimalizací

Počet bodů	1498	5163	14130	49299	118493	172645
t_1 [ms]	378	1209	4308	92628	169299	170710
t_2 [ms]	288	1619	5989	12955	418819	184790
t_3 [ms]	278	1802	5620	37701	61590	481275
t_4 [ms]	468	909	6109	46403	63459	215109
t_5 [ms]	467	1417	9140	22470	77139	171332
$t_ø$ [ms]	**376**	**1391**	**6233**	**42431**	**158061**	**244643**

Lineární spline - s optimalizací

Počet bodů	1498	5163	14130	49299	118493	172645
t_1 [ms]	367	1456	8890	53597	74462	318451
t_2 [ms]	430	1276	11122	48954	114161	252528
t_3 [ms]	331	2272	14003	18848	38485	255473
t_4 [ms]	271	1321	9359	32792	225897	281126
t_5 [ms]	345	2154	4759	26680	69183	445503
$t_ø$ [ms]	**349**	**1696**	**9627**	**36174**	**104438**	**310616**

Tabulka A.1: Rychlost vytváření modelu v závislosti na počtu bodů originálních dat

Akima spline - bez optimalizace

Počet bodů [%]	0,5	1	2,5	5	10	25	50
t_1 [ms]	196	178	199	209	237	277	361
t_2 [ms]	173	189	200	205	221	263	334
t_3 [ms]	173	188	199	208	211	258	345
t_4 [ms]	173	192	198	202	211	278	335
t_5 [ms]	173	190	199	207	211	295	365
$t_ø$ [ms]	**178**	**187**	**199**	**206**	**218**	**274**	**348**

Lineární spline - bez optimalizace

Počet bodů [%]	0,5	1	2,5	5	10	25	50
t_1 [ms]	176	168	158	186	180	177	232
t_2 [ms]	161	159	163	172	174	178	192
t_3 [ms]	157	158	164	165	163	179	195
t_4 [ms]	158	156	183	170	159	180	193
t_5 [ms]	157	157	165	171	160	186	220
$t_ø$ [ms]	**162**	**160**	**167**	**173**	**167**	**180**	**206**

Akima spline - s optimalizací

Počet bodů [%]	0,5	1	2,5	5	10	25	50
t_1 [ms]	11847	26074	21355	109153	112174	791413	2052417
t_2 [ms]	8374	19788	22474	36224	234602	1196493	4006385
t_3 [ms]	13693	25591	20162	61538	299405	982468	5644805
t_4 [ms]	10667	22247	15468	74925	149253	354796	3863469
t_5 [ms]	11296	23172	27014	152794	209670	668830	9431866
$t_ø$ [ms]	**11175**	**23374**	**21295**	**86927**	**201021**	**798800**	**4999788**

Lineární spline - s optimalizací

Počet bodů [%]	0,5	1	2,5	5	10	25	50
t_1 [ms]	12273	25713	68986	44602	246763	1140486	3757732
t_2 [ms]	32371	51290	36567	67362	108982	623268	4151904
t_3 [ms]	29695	11900	26415	48523	191698	636401	4816237
t_4 [ms]	10277	32422	34366	46492	267757	726478	8434699
t_5 [ms]	18713	46943	56930	63381	207512	913487	4234845
$t_ø$ [ms]	**20666**	**33654**	**44653**	**54072**	**204542**	**808024**	**5079083**

Tabulka A.2: Rychlost vytváření modelu v závislosti na počtu bodů tvořících model

Akima spline

Optimalizace	Žádná	1 bodová	2 bodová	Náhodná	Průměrová
t_1 [ms]	193	8795	14805	25895	6507
t_2 [ms]	176	9211	27436	21724	6388
t_3 [ms]	173	10215	23865	13150	6430
t_4 [ms]	174	10914	14574	8695	5580
t_5 [ms]	174	6240	23610	15034	5673
$t_ø$ [ms]	**178**	**9075**	**20858**	**16900**	**6116**

Lineární spline

Optimalizace	Žádná	1 bodová	2 bodová	Náhodná	Průměrová
t_1 [ms]	148	6846	13079	20063	5297
t_2 [ms]	149	8587	23985	12607	4544
t_3 [ms]	145	14378	24191	9394	4868
t_4 [ms]	146	10929	10878	11749	6094
t_5 [ms]	145	8424	8880	14000	6372
$t_ø$ [ms]	**147**	**9833**	**16203**	**13563**	**5435**

Tabulka A.3: Rychlost vytváření modelu v závislosti na typu optimalizace

Akima spline

Ukončení	5	10	25	50	100
t_1 [ms]	4587	30051	46718	56424	254314
t_2 [ms]	13487	14247	14875	48854	99518
t_3 [ms]	15451	11124	16905	54061	73828
t_4 [ms]	4939	11275	56264	34920	188470
t_5 [ms]	10874	16857	48584	111056	69470
$t_{ø}$ [ms]	**9868**	**16711**	**36669**	**61063**	**137120**

Lineární spline

Ukončení	5	10	25	50	100
t_1 [ms]	10483	13152	23238	37012	107877
t_2 [ms]	4525	12371	30551	45385	104681
t_3 [ms]	10821	27291	60096	41011	137971
t_4 [ms]	10928	26443	50510	56649	158351
t_5 [ms]	25545	14909	17292	36992	107629
$t_{ø}$ [ms]	**12460**	**18833**	**36337**	**43410**	**123302**

Tabulka A.4: Rychlost vytváření modelu v závislosti na ukončení

Příloha B – Tabulky s výsledky testování kvality algoritmů na modelování dat

Akima spline - reference 38,8

Ukončení	5	10	25	50	100
$RMSE_1$	34,6	35,3	33,7	32,7	32,8
$RMSE_2$	37,0	32,1	33,2	32,6	32,7
$RMSE_3$	36,3	34,0	34,1	33,4	31,4
$RMSE_4$	33,4	33,6	31,6	32,5	32,5
$RMSE_5$	36,0	32,4	32,9	33,9	33,2
$RMSE_{ø}$	**35,5**	**33,5**	**33,1**	**33,0**	**32,5**
Vylepšení [%]	**8,6**	**13,7**	**14,7**	**14,9**	**16,2**

Lineární spline - reference 37,6

Ukončení	5	10	25	50	100
$RMSE_1$	34,2	33,2	32,2	32,9	30,7
$RMSE_2$	33,8	33,8	33,7	33,2	31,7
$RMSE_3$	33,7	33,1	33,6	32,1	31,4
$RMSE_4$	32,3	32,9	32,2	31,8	33,5
$RMSE_5$	33,4	32,6	31,9	33,7	31,7
$RMSE_{ø}$	**33,5**	**33,1**	**32,7**	**32,7**	**31,8**
Vylepšení [%]	**11,0**	**11,9**	**13,0**	**12,9**	**15,4**

Tabulka B.1: Kvalita modelu v závislosti na ukončení optimalizace

Akima spline - reference 38,8

Optimalizace	1 bodová	2 bodová	Náhodná	Průměrová
$RMSE_1$	35,3	33,6	48,9	36,6
$RMSE_2$	36,0	32,6	49,7	36,2
$RMSE_3$	33,9	34,8	48,8	37,6
$RMSE_4$	34,7	32,6	47,9	36,9
$RMSE_5$	35,0	32,9	48,1	37,2
$RMSE_ø$	**35,0**	**33,3**	**48,7**	**36,9**
Vylepšení [%]	**9,8**	**14,2**	**-25,5**	**4,9**

Lineární spline - reference 37,6

Optimalizace	1 bodová	2 bodová	Náhodná	Průměrová
$RMSE_1$	34,4	32,9	50,9	34,6
$RMSE_2$	34,1	33,2	53,8	36,1
$RMSE_3$	33,5	33,9	56,4	35,2
$RMSE_4$	34,1	33,5	55,6	37,0
$RMSE_5$	34,1	32,1	51,2	35,4
$RMSE_ø$	**34,0**	**33,1**	**53,6**	**35,7**
Vylepšení [%]	**9,5**	**11,9**	**-42,5**	**5,2**

Tabulka B.2: Kvalita modelu v závislosti na typu optimalizace

Akima spline

Vyhlazení	0	10	25	50	100
Reference	**38,8**	**34,5**	**32,9**	**30,8**	**30,3**
$RMSE_1$	34,1	34,3	31,8	30,8	30,8
$RMSE_2$	34,0	32,7	32,5	30,1	30,9
$RMSE_3$	32,2	31,4	30,9	31,1	30,1
$RMSE_4$	34,1	29,4	32,5	30,2	30,2
$RMSE_5$	31,3	31,8	31,5	30,0	30,9
$RMSE_ø$	**33,1**	**31,9**	**31,8**	**30,4**	**30,6**
Vylepšení [%]	**14,6**	**7,5**	**3,2**	**1,2**	**-0,9**

Lineární spline

Vyhlazení	0	10	25	50	100
Reference	**37,6**	**34,1**	**32,6**	**31,0**	**30,6**
$RMSE_1$	34,7	31,5	31,5	31,8	31,4
$RMSE_2$	33,5	32,3	32,3	31,7	31,6
$RMSE_3$	33,0	34,1	32,1	31,9	31,5
$RMSE_4$	33,6	31,3	32,6	32,1	31,6
$RMSE_5$	35,2	31,8	32,3	32,4	31,6
$RMSE_ø$	**34,0**	**32,2**	**32,2**	**32,0**	**31,5**
Vylepšení [%]	**9,6**	**5,6**	**1,3**	**-3,2**	**-3,1**

Tabulka B.3: Kvalita modelu v závislosti na vyhlazení dat

Příloha C – Tabulky s výsledky testování rychlosti vyhledávacích algoritmů

Metoda délky vektoru

Počet bodů	1498	14130	49299	118493	172645
t_1 [ms]	97	1498	11744	59380	123545
t_2 [ms]	92	1502	11712	59399	125245
t_3 [ms]	87	1495	11747	59975	122843
t_4 [ms]	90	1504	11699	59609	125125
t_5 [ms]	93	1510	11697	59732	123325
$t_ø$ [ms]	**92**	**1502**	**11720**	**59619**	**124017**

Metoda nejbližších sousedů

Počet bodů	1498	14130	49299	118493	172645
t_1 [ms]	44	1000	9979	55179	117326
t_2 [ms]	37	989	10015	55015	115651
t_3 [ms]	41	1004	9979	55077	116081
t_4 [ms]	33	1001	9966	54982	115347
t_5 [ms]	31	994	9983	55436	116441
$t_ø$ [ms]	**37**	**998**	**9984**	**55138**	**116169**

Tabulka C.1: Rychlost vyhledávání v závislosti na počtu bodů originálních dat

Metoda délky vektoru

Počet bodů	100	500	1000	5000	10000
t_1 [ms]	11744	20717	32027	115600	198833
t_2 [ms]	11726	20938	31951	116040	197933
t_3 [ms]	11713	20807	31962	118504	198567
t_4 [ms]	11697	20723	31886	118159	199362
t_5 [ms]	11776	20734	32031	118913	203674
$t_ø$ [ms]	**11731**	**20784**	**31971**	**117443**	**199674**

Metoda nejbližších sousedů

Počet bodů	100	500	1000	5000	10000
t_1 [ms]	9984	12698	16295	43187	72445
t_2 [ms]	10032	12685	16279	42987	72197
t_3 [ms]	10124	12682	16264	42983	72035
t_4 [ms]	10013	12707	16346	42977	71406
t_5 [ms]	9998	12685	16411	43152	71171
$t_ø$ [ms]	**10030**	**12691**	**16319**	**43057**	**71851**

Tabulka C.2: Rychlost vyhledávání v závislosti na délce hledaného úseku

Porovnávání dnů - 1440 vzorků/den

Metoda	Pear. korelace	Sign. korelace	RMSE	Vše
t_1 [ms]	1965	579	521	3040
t_2 [ms]	1958	575	516	3054
t_3 [ms]	1960	579	511	3045
t_4 [ms]	1964	578	512	3052
t_5 [ms]	1955	578	520	3052
$t_ø$ [ms]	**1960**	**578**	**516**	**3049**

Tabulka C.3: Rychlost porovnávání v závislosti na zvolené metodě